NI-488.2™
User Manual for Windows

January 1996 Edition

Part Number 370902A-01

© Copyright 1993, 1996 National Instruments Corporation.
All Rights Reserved.

National Instruments Corporate Headquarters
6504 Bridge Point Parkway
Austin, TX 78730-5039
(512) 794-0100
Technical support fax: (800) 328-2203
(512) 794-5678

Branch Offices:
Australia 03 9 879 9422, Austria 0662 45 79 90 0, Belgium 02 757 00 20,
Canada (Ontario) 519 622 9310, Canada (Québec) 514 694 8521,
Denmark 45 76 26 00, Finland 90 527 2321, France 1 48 14 24 24,
Germany 089 741 31 30, Hong Kong 2645 3186, Italy 02 48301892,
Japan 03 5472 2970, Korea 02 596 7456, Mexico 95 800 010 0793,
Netherlands 0348 433466, Norway 32 84 84 00, Singapore 2265886,
Spain 91 640 0085, Sweden 08 730 49 70, Switzerland 056 200 51 51,
Taiwan 02 377 1200, U.K. 01635 523545

Limited Warranty

The media on which you receive National Instruments software are warranted not to fail to execute programming instructions, due to defects in materials and workmanship, for a period of 90 days from date of shipment, as evidenced by receipts or other documentation. National Instruments will, at its option, repair or replace software media that do not execute programming instructions if National Instruments receives notice of such defects during the warranty period. National Instruments does not warrant that the operation of the software shall be uninterrupted or error free.

A Return Material Authorization (RMA) number must be obtained from the factory and clearly marked on the outside of the package before any equipment will be accepted for warranty work. National Instruments will pay the shipping costs of returning to the owner parts which are covered by warranty.

National Instruments believes that the information in this manual is accurate. The document has been carefully reviewed for technical accuracy. In the event that technical or typographical errors exist, National Instruments reserves the right to make changes to subsequent editions of this document without prior notice to holders of this edition. The reader should consult National Instruments if errors are suspected. In no event shall National Instruments be liable for any damages arising out of or related to this document or the information contained in it.

EXCEPT AS SPECIFIED HEREIN, NATIONAL INSTRUMENTS MAKES NO WARRANTIES, EXPRESS OR IMPLIED, AND SPECIFICALLY DISCLAIMS ANY WARRANTY OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. CUSTOMER'S RIGHT TO RECOVER DAMAGES CAUSED BY FAULT OR NEGLIGENCE ON THE PART OF NATIONAL INSTRUMENTS SHALL BE LIMITED TO THE AMOUNT THERETOFORE PAID BY THE CUSTOMER. NATIONAL INSTRUMENTS WILL NOT BE LIABLE FOR DAMAGES RESULTING FROM LOSS OF DATA, PROFITS, USE OF PRODUCTS, OR INCIDENTAL OR CONSEQUENTIAL DAMAGES, EVEN IF ADVISED OF THE POSSIBILITY THEREOF. This limitation of the liability of National Instruments will apply regardless of the form of action, whether in contract or tort, including negligence. Any action against National Instruments must be brought within one year after the cause of action accrues. National Instruments shall not be liable for any delay in performance due to causes beyond its reasonable control. The warranty provided herein does not cover damages, defects, malfunctions, or service failures caused by owner's failure to follow the National Instruments installation, operation, or maintenance instructions; owner's modification of the product; owner's abuse, misuse, or negligent acts; and power failure or surges, fire, flood, accident, actions of third parties, or other events outside reasonable control.

Copyright

Under the copyright laws, this publication may not be reproduced or transmitted in any form, electronic or mechanical, including photocopying, recording, storing in an information retrieval system, or translating, in whole or in part, without the prior written consent of National Instruments Corporation.

Trademarks

NI-488®, NI-488.2™, and TNT4882C™ are trademarks of National Instruments Corporation.

Product and company names listed are trademarks or trade names of their respective companies.

WARNING REGARDING MEDICAL AND CLINICAL USE OF NATIONAL INSTRUMENTS PRODUCTS

National Instruments products are not designed with components and testing intended to ensure a level of reliability suitable for use in treatment and diagnosis of humans. Applications of National Instruments products involving medical or clinical treatment can create a potential for accidental injury caused by product failure, or by errors on the part of the user or application designer. Any use or application of National Instruments products for or involving medical or clinical treatment must be performed by properly trained and qualified medical personnel, and all traditional medical safeguards, equipment, and procedures that are appropriate in the particular situation to prevent serious injury or death should always continue to be used when National Instruments products are being used. National Instruments products are NOT intended to be a substitute for any form of established process, procedure, or equipment used to monitor or safeguard human health and safety in medical or clinical treatment.

Contents

About This Manual xiii
How to Use This Manual Set xiii
Organization of This Manual xiv
Conventions Used in This Manual xv
Related Documentation xvi
Customer Communication xvi

Chapter One
Introduction 1-1
GPIB Overview 1-1
Talkers, Listeners, and Controllers 1-1
Controller-In-Charge and System Controller 1-1
GPIB Addressing 1-2
Sending Messages Across the GPIB 1-2
Data Lines 1-2
Handshake Lines 1-3
Interface Management Lines 1-3
Setting Up and Configuring Your System 1-4
Controlling More Than One Board 1-5
Configuration Requirements 1-5
The NI-488.2 Software Package 1-6
NI-488.2 Driver and Driver Utilities 1-6
C Language Files 1-7
Visual Basic Language Files 1-7
Win 32s Files 1-8
Sample Application Files 1-8
How the NI-488.2 Software Works with Windows 1-8
Unloading and Reloading the NI-488.2 Driver 1-9

Chapter 2
Application Examples 2-1
Example 1: Basic Communication 2-2
Example 2: Clearing and Triggering Devices 2-4
Example 3: Asynchronous I/O 2-6
Example 4: End-of-String Mode 2-8
Example 5: Service Requests 2-10
Example 6: Basic Communication with IEEE 488.2-Compliant Devices 2-14
Example 7: Serial Polls Using NI-488.2 Routines 2-16
Example 8: Parallel Polls 2-18
Example 9: Non-Controller Example 2-20

Chapter 3
Developing Your Application 3-1

Choosing How to Access the NI-488.2 DLL 3-1
Choosing Between NI-488 Functions and NI-488.2 Routines 3-1
Using NI-488 Functions: One Device for Each Board 3-2
NI-488 Device Functions 3-2
NI-488 Board Functions 3-2
Using NI-488.2 Routines: Multiple Boards and/or Multiple Devices 3-3
Checking Status with Global Variables 3-3
Status Word – ibsta 3-3
Error Variable – iberr 3-5
Count Variables – ibcnt and ibcntl 3-5
Using wibic to Communicate with Devices 3-5
Writing Your NI-488 Application 3-6
Items to Include 3-6
NI-488 Program Shell 3-7
General Program Steps and Examples 3-8
Step 1. Open a Device 3-8
Step 2. Clear the Device 3-8
Step 3. Configure the Device 3-9
Step 4. Trigger the Device 3-9
Step 5. Wait for the Measurement 3-9
Step 6. Read the Measurement 3-10
Step 7. Process the Data 3-10
Step 8. Place the Device Offline 3-10
Writing Your NI-488.2 Application 3-11
Items to Include 3-11
NI-488.2 Program Shell 3-12
General Program Steps and Examples 3-13
Step 1. Initialization 3-13
Step 2. Find All Listeners 3-13
Step 3. Identify the Instrument 3-13
Step 4. Initialize the Instrument 3-14
Step 5. Configure the Instrument 3-15
Step 6. Trigger the Instrument 3-15
Step 7. Wait for the Measurement 3-15
Step 8. Read the Measurement 3-16
Step 9. Process the Data 3-16
Step 10. Place the Board Offline 3-16
Compiling, Linking, and Running Your Application 3-17
Microsoft C 3-17
Borland C++ 3-17
Visual Basic 3-18
Direct Entry with C 3-18
Method 1. List Functions in the Module Definition File 3-19

Method 2. Generate an Import Library Using implib 3-20
Method 3. Use LoadLibrary and GetProcAddress 3-20
Direct Entry with Visual Basic 3-23

Chapter 4
Debugging Your Application 4-1
Running wibtest 4-1
Presence Test of Driver 4-1
Presence Test of Board 4-1
GPIB Cables Connected 4-2
Running GPIBInfo 4-2
Debugging with the Global Status Variables 4-3
Debugging with wibic 4-4
Debugging with GPIB Spy 4-4
GPIB Error Codes 4-4
Configuration Errors 4-5
Timing Errors 4-6
Communication Errors 4-6
Repeat Addressing 4-6
Termination Method 4-6
Common Questions 4-7

Chapter 5
wibic–Windows Interface Bus Interactive Control Utility 5-1
Overview 5-1
Example Using NI-488 Functions 5-1
wibic Syntax 5-4
Number Syntax 5-4
String Syntax 5-5
Address Syntax 5-5
wibic Syntax for NI-488 Functions 5-5
wibic Syntax for NI-488.2 Routines 5-8
Status Word 5-9
Error Information 5-9
Count 5-10
Common NI-488 Functions 5-10
ibfind 5-10
ibdev 5-10
ibwrt 5-12
ibrd 5-12
Common NI-488.2 Routines in wibic 5-13
Set 5-13
Send and SendList 5-13
Receive 5-13
Auxiliary Functions 5-14
Set (Select Device or Board) 5-15
Help (Display Help Information) 5-15

! (Repeat Previous Function) 5-15
- (Turn OFF Display) and + (Turn ON Display) 5-15
n* (Repeat Function n Times) 5-16
$ (Execute Indirect File) 5-16
Print (Display the ASCII String) 5-17
Buffer (Set Buffer Display Mode) 5-17

Chapter 6
GPIB Spy 6-1
Overview 6-1
Starting GPIB Spy 6-1
Exiting GPIB Spy 6-2
Viewing Call Details 6-2
GPIB Spy Configuration Options 6-3
Calls to Highlight 6-3
Calls To Trap On 6-3
Buffer History 6-3
Set Highlight Color 6-3
Store Configuration 6-4
GPIB Spy Output Options 6-4
To Screen 6-4
To File 6-4
Clear Screen 6-4
GPIB Spy Help 6-5
Performance Considerations 6-5

Chapter 7
GPIB Programming Techniques 7-1
Termination of Data Transfers 7-1
High-Speed Data Transfers (HS488) 7-2
Enabling HS488 7-2
System Configuration Effects on HS488 7-3
Waiting for GPIB Conditions 7-3
Device-Level Calls and Bus Management 7-3
Talker/Listener Applications 7-4
Waiting for Messages from the Controller 7-4
Using the Event Queue 7-5
Requesting Service 7-5
Serial Polling 7-5
Service Requests from IEEE 488 Devices 7-5
Service Requests from IEEE 488.2 Devices 7-6
Automatic Serial Polling 7-6
Stuck SRQ State 7-6
Autopolling and Interrupts 7-7
SRQ and Serial Polling with NI-488 Device Functions 7-7
SRQ and Serial Polling with NI-488.2 Routines 7-8
Example 1: Using FindRQS 7-9
Example 2: Using AllSpoll 7-9

Parallel Polling 7-10
Implementing a Parallel Poll 7-10
Parallel Polling with NI-488 Functions 7-10
Parallel Polling with NI-488.2 Routines 7-12

Chapter 8
wibconf–Windows Interface Bus Configuration Utility 8-1
Overview 8-1
Starting wibconf 8-1
Upper and Lower Levels of wibconf 8-2
Upper Level Device Map 8-3
Device Maps of the Boards 8-4
Help 8-4
Rename 8-4
(Dis)connect 8-4
Edit 8-5
Exit 8-5
Lower Level Device/Board Characteristics 8-5
Change Characteristics 8-6
Change Board or Device 8-6
Help 8-6
Reset Value 8-6
Return to Map 8-6
Board and Device Configuration Options 8-7
Primary GPIB Address 8-7
Secondary GPIB Address 8-7
Timeout Setting 8-7
Serial Poll Timeouts (Device Characteristic Only) 8-8
Terminate Read on EOS 8-8
Set EOI with EOS on Writes 8-8
Type of Compare on EOS 8-8
EOS Byte 8-8
Send EOI at End of Write 8-9
System Controller (Board Characteristic Only) 8-9
Assert REN when SC (Board Characteristic Only) 8-9
Enable Auto Serial Polling (Board Characteristic Only) 8-9
Enable CIC Protocol (Board Characteristic Only) 8-9
Bus Timing (Board Characteristic Only) 8-10
Cable Length for High Speed (Board Characteristic Only) 8-10
Parallel Poll Duration (Board Characteristic Only) 8-10
Use This GPIB Interface (Board Characteristic Only) 8-10
Base I/O Address (Board Description Only) 8-10
DMA Channel (Board Characteristic Only) 8-11
Interrupt Jumper Setting (Board Characteristic Only) 8-11
Serial Poll Timeout (Device Characteristic Only) 8-11
Enable Repeat Addressing (Device Characteristic Only) 8-11
GPIB-PCII/IIA Mode Switch 8-11
Default Configurations in wibconf 8-12

Exiting wibconf 8-12
Checking for Errors 8-13
wibconf Batch Mode 8-13

Appendix A
Status Word Conditions A-1

Appendix B
Error Codes and Solutions B-1

Appendix C
Customer Communication C-1
Glossary Glossary-1

Index Index-1

Figures

Figure 1-1. GPIB Address Bits 1-2
Figure 1-2. Linear and Star System Configuration 1-4
Figure 1-3. Example of Multiboard System Setup 1-5
Figure 1-4. How the NI-488.2 Software Works with Windows 1-8

Figure 2-1. Program Flowchart for Example 1 2-3
Figure 2-2. Program Flowchart for Example 2 2-5
Figure 2-3. Program Flowchart for Example 3 2-7
Figure 2-4. Program Flowchart for Example 4 2-9
Figure 2-5. Program Flowchart for Example 5 2-12
Figure 2-6. Program Flowchart for Example 6 2-15
Figure 2-7. Program Flowchart for Example 7 2-17
Figure 2-8. Program Flowchart for Example 8 2-19
Figure 2-9. Program Flowchart for Example 9 2-21

Figure 3-1. General Program Shell Using NI-488 Device Functions 3-7
Figure 3-2. General Program Shell Using NI-488.2 Routines 3-12

Figure 6-1. Main GPIB Spy Window 6-1
Figure 6-2 GPIB Spy Call Details Window for ibwrt 6-2

Figure 8-1. Upper Level of wibconf 8-3
Figure 8-2. Lower Level of wibconf 8-5

Tables

Table 1-1. GPIB Handshake Lines 1-3
Table 1-2. GPIB Interface Management Lines 1-3

Table 3-1. Status Word (ibsta) Layout 3-4

Table 4-1. GPIB Error Codes 4-5

Table 5-1. Syntax for Device-Level NI-488 Functions in wibic..... 5-6
Table 5-2. Syntax for Board-Level NI-488 Functions in wibic 5-7
Table 5-3. Syntax for NI-488.2 Routines in wibic 5-8
Table 5-4. Auxiliary Functions in wibic 5-14

Table 8-1. Options for Starting wibconf 8-2
Table 8-2. wibconf Batch Mode Command Pairs..... 8-15

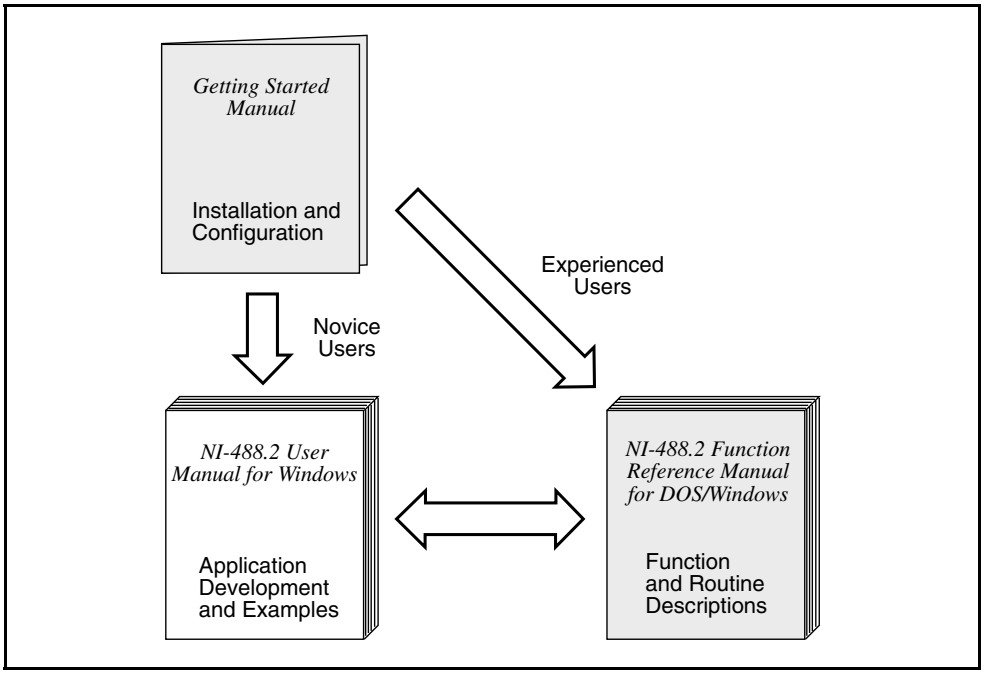
Table A-1. Status Word Bits A-1

Table B-1. GPIB Error Codes B-1

About This Manual

This manual describes the features and functions of the NI-488.2 software for Windows. The NI-488.2 software is meant to be used with Microsoft Windows version 3.0 or higher. This manual assumes that you are already familiar with Windows.

How to Use This Manual Set



Use the getting started manual to install and configure your GPIB hardware and NI-488.2 software for Windows.

Use the *NI-488.2 User Manual for Windows* to learn the basics of GPIB and how to develop an application program. The user manual also contains debugging information and detailed examples.

Use the *NI-488.2 Function Reference Manual for DOS/Windows* for specific information about each NI-488 function and NI-488.2 routine such as format, parameters, and possible errors.

Organization of This Manual

This manual is organized as follows:

- Chapter 1, *Introduction*, gives an overview of GPIB and the NI-488.2 software.
- Chapter 2, *Application Examples*, contains nine sample applications designed to illustrate specific GPIB concepts and techniques that can help you write your own applications. The description of each example includes the programmer's task, a program flowchart, and numbered steps which correspond to the numbered blocks on the flowchart.
- Chapter 3, *Developing Your Application,* explains how to develop a GPIB application program using NI-488 functions and NI-488.2 routines.
- Chapter 4, *Debugging Your Application,* describes several ways to debug your application program.
- Chapter 5, *wibic–Windows Interface Bus Interactive Control*, introduces you to `wibic`, the interactive control program that you can use to communicate with GPIB devices interactively.
- Chapter 6, *GPIB Spy*, explains how to use the GPIB Spy utility to monitor and debug your application program.
- Chapter 7, *GPIB Programming Techniques*, describes techniques for using some NI-488 functions and NI-488.2 routines in your application program.
- Chapter 8, *wibconf–Windows Interface Bus Configuration Utility*, contains a description of `wibconf`, the software configuration program you can use to configure the NI-488.2 software for Windows.
- Appendix A, *Status Word Conditions*, gives a detailed description of the conditions reported in the status word, `ibsta`.
- Appendix B, *Error Codes and Solutions*, lists a description of each error, some conditions under which it might occur, and possible solutions.
- Appendix C, *Customer Communication*, contains forms you can use to request help from National Instruments or to comment on our products and manuals.
- The *Glossary* contains an alphabetical list and description of terms used in this manual, including abbreviations, acronyms, metric prefixes, mnemonics, and symbols.
- The *Index* contains an alphabetical list of key terms and topics in this manual, including the page where you can find each one.

Conventions Used in This Manual

The following conventions are used in this manual.

bold	Text in bold denotes windows, menus, menu options, and dialog box options.
bold italic	Bold italic text denotes a note, caution, or warning.
italic	Italic text denotes emphasis, cross references, field names, or an introduction to a key concept.
`monospace`	Text in this font denotes text or characters that you enter from the keyboard. Sections of code, programming examples, and syntax examples also appear in this font. This font is also used for the proper name of disk drives, paths, directories, device names, variables, and for statements taken from program code.
`bold monospace`	Bold text in this font denotes the messages and responses that the computer automatically prints to the screen.
`italic monospace`	Italic text in this font denotes that you must supply the appropriate words or values in the place of these items.
<>	Angle brackets enclose the name of a key on the keyboard–for example, <PageDown>.
-	A hyphen between two or more key names enclosed in angle brackets denotes that you should simultaneously press the named keys–for example, <Control-C>.
IEEE 488 and IEEE 488.2	*IEEE 488* and *IEEE 488.2* are used throughout this manual to refer to the ANSI/IEEE Standard 488.1-1987 and the ANSI/IEEE Standard 488.2-1992, respectively, which define the GPIB.
NI-488.2 software	The term *NI-488.2 software* is used throughout this manual to refer to the NI-488.2 software for Windows, unless otherwise noted.

Abbreviations, acronyms, metric prefixes, mnemonics, symbols, and terms are listed in the *Glossary*.

Related Documentation

The following documents contain information that you may find helpful as you read this manual:

- ANSI/IEEE Standard 488.1-1987, *IEEE Standard Digital Interface for Programmable Instrumentation*
- ANSI/IEEE Standard 488.2-1992, *IEEE Standard Codes, Formats, Protocols, and Common Commands*
- *Microsoft Windows User's Guide*

Customer Communication

National Instruments wants to receive your comments on our products and manuals. We are interested in the applications you develop with our products, and we want to help if you have problems with them. To make it easy for you to contact us, this manual contains comment and configuration forms for you to complete. These forms are in Appendix C, *Customer Communication*, at the end of this manual.

Chapter 1
Introduction

This chapter gives an overview of GPIB and the NI-488.2 software.

GPIB Overview

The ANSI/IEEE Standard 488.1-1987, also known as GPIB (General Purpose Interface Bus), describes a standard interface for communication between instruments and controllers from various vendors. It contains information about electrical, mechanical, and functional specifications. The GPIB is a digital, 8-bit parallel communications interface with data transfer rates of 1 Mbytes/s and above. The bus supports one System Controller, usually a computer, and up to 14 additional instruments. The ANSI/IEEE Standard 488.2-1987 extends IEEE 488.1 by defining a bus communication protocol, a common set of data codes and formats, and a generic set of common device commands.

Talkers, Listeners, and Controllers

GPIB devices can be Talkers, Listeners, or Controllers. A Talker sends out data messages. Listeners receive data messages. The Controller, usually a computer, manages the flow of information on the bus. It defines the communication links and sends GPIB commands to devices.

Some devices are capable of playing more than one role. A digital voltmeter, for example, can be a Talker and a Listener. If your personal computer has a National Instruments GPIB interface board and NI-488.2 software installed, it can function as a Talker, Listener, and Controller.

Controller-In-Charge and System Controller

You can have multiple Controllers on the GPIB, but only one Controller at a time can be the active Controller, or Controller-In-Charge (CIC). The CIC can either be active or inactive (Standby) Controller. Control can pass from the current CIC to an idle Controller, but only the System Controller, usually a GPIB interface board, can make itself the CIC.

GPIB Addressing

All GPIB devices and boards must be assigned a unique GPIB address. A GPIB address is made up of two parts: a primary address and an optional secondary address.

The primary address is a number in the range 0 to 30. The GPIB Controller uses this address to form a talk or listen address that is sent over the GPIB when communicating with a device.

A talk address is formed by setting bit 6, the TA (Talk Active) bit of the GPIB address. A listen address is formed by setting bit 5, the LA (Listen Active) bit of the GPIB address. For example, if a device is at address 1, the Controller sends hex 41 (address 1 with bit 6 set) to make the device a Talker. Because the Controller is usually at primary address 0, it sends hex 20 (address 0 with bit 5 set) to make itself a Listener. Figure 1-1 shows the configuration of the GPIB address bits.

Bit Position	7	6	5	4	3	2	1	0
Meaning	0	TA	LA	GPIB Primary Address (range 0 to 30)				

Figure 1-1. GPIB Address Bits

With some devices, you can use secondary addressing. A secondary address is a number in the range hex 60 to hex 7E. When secondary addressing is in use, the Controller sends the primary talk or listen address of the device followed by the secondary address of the device.

Sending Messages Across the GPIB

Devices on the bus communicate by sending messages. Signals and lines transfer these messages across the GPIB interface, which consists of 16 signal lines and eight ground return (shield drain) lines. The 16 signal lines are discussed in the following sections.

Data Lines

Eight data lines, DIO1 through DIO8, carry both data and command messages.

Handshake Lines

Three hardware handshake lines asynchronously control the transfer of message bytes between devices. This process is a three-wire interlocked handshake, and it guarantees that devices send and receive message bytes on the data lines without transmission error. Table 1-1 summarizes the GPIB handshake lines.

Table 1-1. GPIB Handshake Lines

Line	Description
NRFD (not ready for data)	Listening device is ready/not ready to receive a message byte. Also used by the Talker to signal high-speed GPIB transfers.
NDAC (not data accepted)	Listening device has/has not accepted a message byte.
DAV (data valid)	Talking device indicates signals on data lines are stable (valid) data.

Interface Management Lines

Five GPIB hardware lines manage the flow of information across the bus. Table 1-2 summarizes the GPIB interface management lines.

Table 1-2. GPIB Interface Management Lines

Line	Description
ATN (attention)	Controller drives ATN true when it sends commands and false when it sends data messages.
IFC (interface clear)	System Controller drives the IFC line to initialize the bus and make itself CIC.
REN (remote enable)	System Controller drives the REN line to place devices in remote or local program mode.
SRQ (service request)	Any device can drive the SRQ line to asynchronously request service from the Controller.
EOI (end or identify)	Talker uses the EOI line to mark the end of a data message. Controller uses the EOI line when it conducts a parallel poll.

Setting Up and Configuring Your System

Devices are usually connected with a cable assembly consisting of a shielded 24-conductor cable with both a plug and receptacle connector at each end. With this design, you can link devices in a linear configuration, a star configuration, or a combination of the two. Figure 1-2 shows the linear and star configurations.

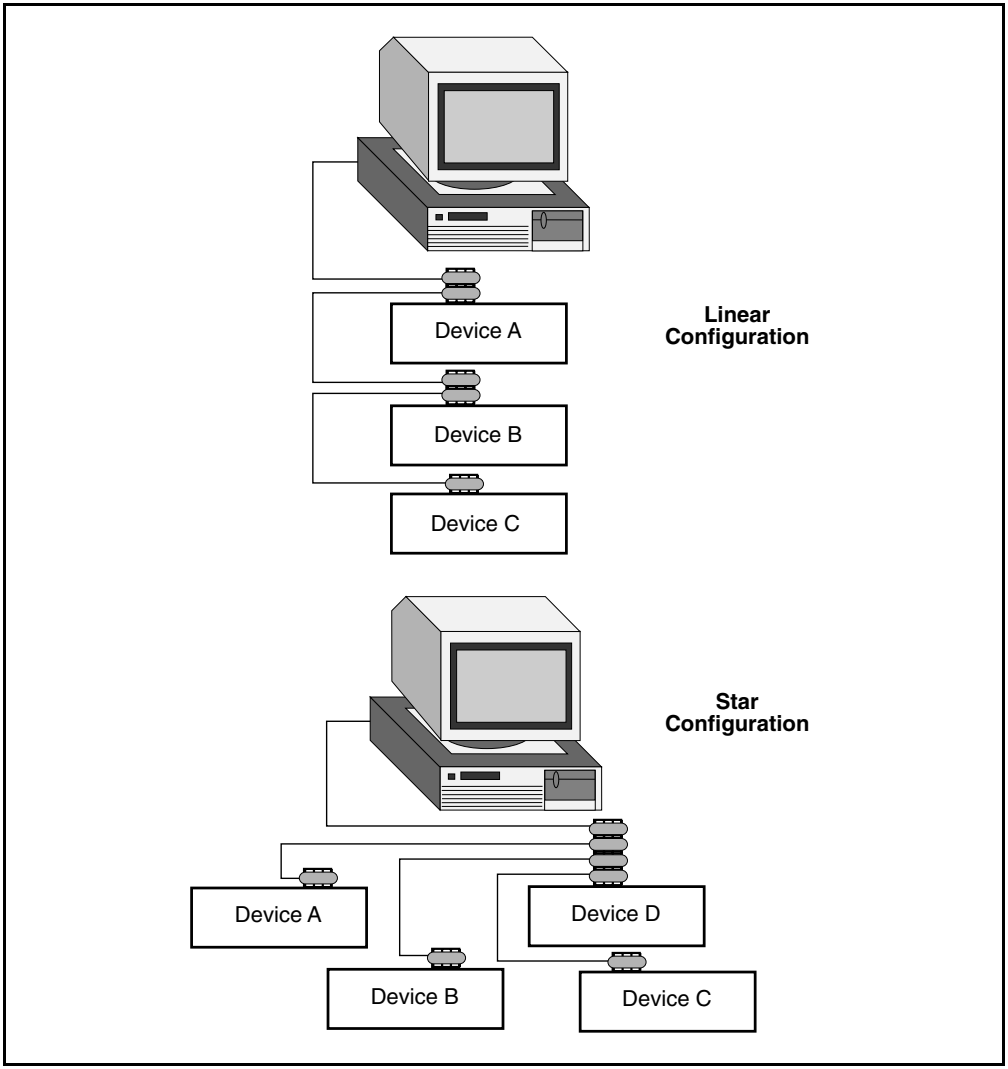


Figure 1-2. Linear and Star System Configuration

Controlling More Than One Board

Multiboard drivers, such as the NI-488.2 driver for Windows, can control more than one interface board. Figure 1-3 shows an example of a multiboard system configuration. `gpib0` is the access board for the voltmeter, and `gpib1` is the access board for the plotter and printer. The control functions of the devices automatically access their respective boards.

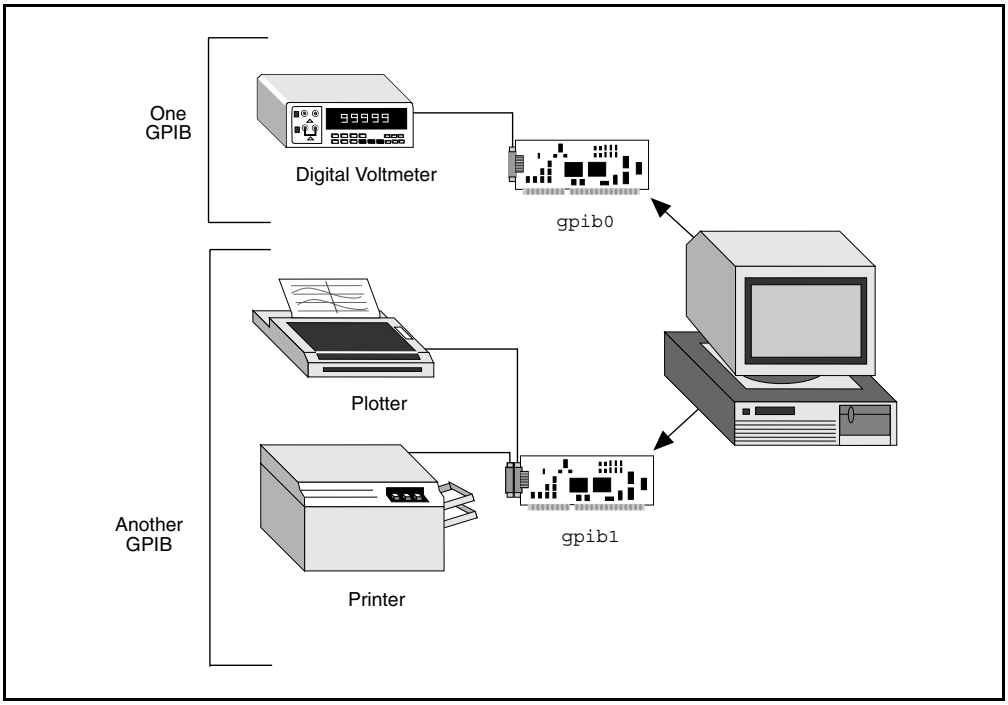


Figure 1-3. Example of Multiboard System Setup

Configuration Requirements

To achieve the high data transfer rate that the GPIB was designed for, you must limit the physical distance between devices and the number of devices on the bus. The following restrictions are typical:

- A maximum separation of four meters between any two devices and an average separation of two meters over the entire bus.
- A maximum total cable length of 20 m.
- A maximum of 15 devices connected to each bus, with at least two-thirds powered on.

For high-speed operation, the following restrictions apply:

- All devices in the system must be powered on.
- Cable lengths as short as possible up to a maximum of 15 m of cable for each system.
- At least one equivalent device load per meter of cable.

If you want to exceed these limitations, you can use bus extenders to increase the cable length or expander to increase the number of device loads. Extenders and expanders are available from National Instruments.

The following sections describe the NI-488.2 software, which controls the flow of communication on the GPIB.

The NI-488.2 Software Package

The following section highlights important elements of the NI-488.2 software for Windows and describes the function of each element.

NI-488.2 Driver and Driver Utilities

The NI-488.2 software contains the following driver and utility files:

- `readme.txt` is a documentation file that contains important information about the NI-488.2 software and a description of any new features. Before you use the software, read this file for the most recent information.
- `setup.exe` is a Windows setup executable that installs the NI-488.2 software.
- `gpib.dll` is a 16-bit dynamic link library that implements the NI-488 and NI-488.2 functions.
- `gpib.ini` is the GPIB configuration information file that describes the default configuration of the NI-488.2 software.
- `nivgpibd.386`, the National Instruments virtual GPIB device, allows you to use DMA to transfer buffers located above 16 MB in memory. The setup program offers to install this virtual device for you.
- `nivdmad.386` is an improved version of the virtual DMA device (`vdmad`) used by Windows 3.0 in 386-enhanced mode. The setup program offers to install this virtual device for you if you are running Windows 3.0.

- `ibdiag.exe` is a program that you can use to test the GPIB hardware and ensure that it is functioning properly.
- `wibtest.exe` is the Windows NI-488.2 software installation test.
- `gpibinfo.exe` is a utility you can use to obtain information about your GPIB hardware and software, such as the version of the NI-488.2 software and the type of interface board you are using.
- `wibic.exe` is an interactive control program that you use to communicate with the GPIB devices interactively using NI-488.2 functions and routines. It helps you to learn the NI-488.2 routines and to program your instrument or other GPIB devices.
- `wibconf.exe` is a software configuration program that changes the configuration parameters of the NI-488.2 software for Windows.
- `gpibspy.exe` is the GPIB activity monitor program. It is a debugging tool that you can use to monitor the NI-488.2 calls made by your Windows applications.

C Language Files

The NI-488.2 software contains the following C language files:

- `readme.mc` is a documentation file that contains information about the C language interface.
- `gpib.lib` is the Microsoft C (version 5.1 or higher) and Borland C++ (version 2.0 or higher) language interface library. You must link this library with your application program so that your program can access the NI-488.2 dynamic link library.
- `windecl.h` is an include file. It contains NI-488 function and NI-488.2 routine prototypes and various predefined constants.

Visual Basic Language Files

The distribution disk contains the following language files for Microsoft Visual Basic:

- `readme.vb` is a documentation file that contains information about the Visual Basic language interface.
- `vbib.bas` is the Visual Basic (version 1.0 or higher) language source file which defines the NI-488 functions and NI-488.2 routines that call the `gpib.dll`.
- `niglobal.bas` is the Visual Basic global module that includes certain predefined constant declarations and declares the global variables.

Win 32s Files

- `gpib-32.dll` is a 32-bit translation dynamic link library that handles the 32-bit to 16-bit translation.
- `gpib-16.dll` is a 16-bit translation dynamic link library.
- `wibic32.exe` is a Win32s version of the `wibic` application.

Sample Application Files

The NI-488.2 software includes nine sample applications along with source code for each language supported by the NI-488.2 software. For a detailed description of the sample application files, refer to Chapter 2, *Application Examples*.

How the NI-488.2 Software Works with Windows

The NI-488.2 driver is a standard 16-bit Windows DLL, which is loaded when a Windows GPIB application is executed. The NI-488.2 driver for Windows is unloaded when the last active GPIB application is exited.

Figure 1-4 shows how the NI-488.2 software works with Windows and your GPIB hardware.

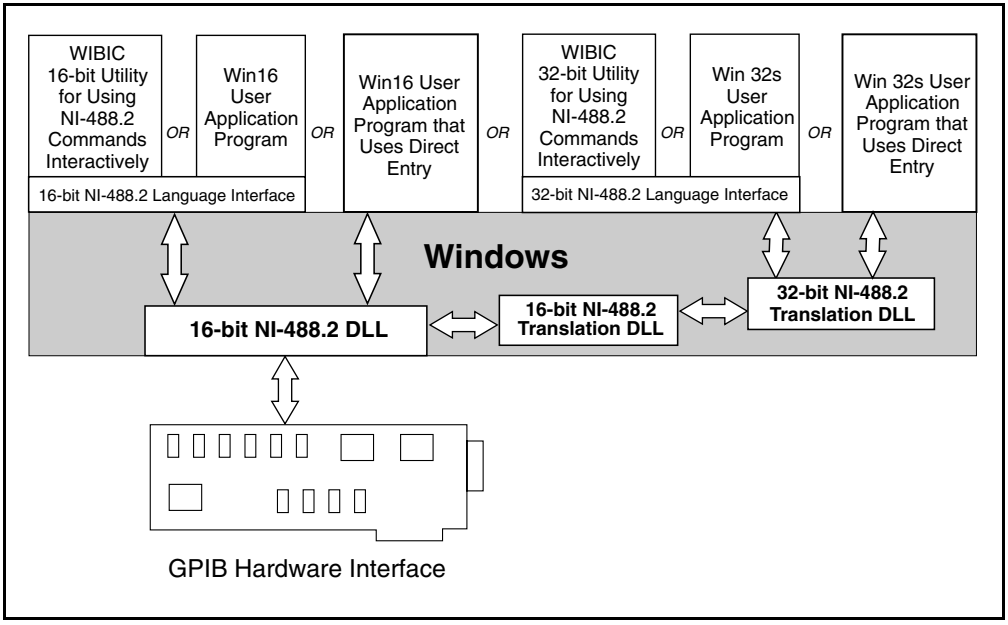


Figure 1-4. How the NI-488.2 Software Works with Windows

Unloading and Reloading the NI-488.2 Driver

The NI-488.2 driver for Windows is a dynamic link library (DLL), which is loaded and unloaded automatically. When a Windows application first accesses the NI–488.2 DLL, it loads the DLL into memory. When the last Windows application using the NI-488.2 DLL has exited, the DLL is unloaded from memory.

The NI-488.2 DLL might use the services of the virtual device driver, `nivgpibd.386`. If you have more than 16 MB of RAM in your computer and you use DMA to read from and write to your GPIB instrument, then you must have the `nivgpibd.386` device installed. If it is not installed, the EDMA error might be returned.

You can remove the `nivgpibd.386` device by editing the `system.ini` file as follows:

1. Locate the `system.ini` file in the Windows directory (usually `c:\windows`).
2. Find the line in the [`386Enh`] section that is of the following form:

   ```
   device = drive:\path\nivgpibd.386
   ```

   where `drive` is the drive (usually `c`) and `path` is the path where the NI-488.2 software is installed (for example, `c:\at-gpibw`).
3. Change this command line to a comment by adding a semicolon at the beginning of the line as follows:

   ```
   ;device = drive:\path\nivgpibd.386
   ```
4. Restart Windows so that the change in `system.ini` can take effect.

Chapter 2
Application Examples

This chapter contains nine sample applications designed to illustrate specific GPIB concepts and techniques that can help you write your own applications. The description of each example includes the programmer's task, a program flowchart, and numbered steps which correspond to the numbered blocks on the flowchart.

Use this chapter along with your NI-488.2 software, which includes the C and Visual Basic source code for each of the nine examples. The programs are listed in order of increasing complexity. If you are new to GPIB programming, you might want to study the contents and concepts of the first sample, `simple.c`, before moving on to more complex examples.

The following example programs are included with your NI-488.2 software:

- `simple.c` is the source code file for Example 1. It illustrates how you can establish communication between a host computer and a GPIB device.
- `clr_trg.c` is the source code file for Example 2. It illustrates how you can clear and trigger GPIB devices.
- `asynch.c` is the source code file for Example 3. It illustrates how you can perform non-GPIB tasks while data is being transferred over the GPIB.
- `eos.c` is the source code file for Example 4. It illustrates the concept of the end-of-string (EOS) character.
- `rqs.c` is the source code file for Example 5. It illustrates how you can communicate with GPIB devices that use the GPIB SRQ line to request service. This sample is written using NI-488 functions.
- `easy4882.c` is the source code file for Example 6. It is an introduction to NI-488.2 routines.
- `rqs4882.c` is the source code file for Example 7. It uses NI-488.2 routines to communicate with GPIB devices that use the GPIB SRQ line to request service.
- `ppoll.c` is the source code file for Example 8. It uses NI-488.2 routines to conduct parallel polls.
- `non_cic.c` is the source code file for Example 9. It illustrates how you can use the NI-488.2 driver in a non-Controller application.

Example 1: Basic Communication

This example focuses on the basics of establishing communication between a host computer and a GPIB device.

A technician needs to monitor voltage readings using a GPIB multimeter. His computer is equipped with an IEEE 488.2 interface board. The NI-488.2 software is installed, and a GPIB cable runs from the computer to the GPIB port on the multimeter.

The technician is familiar with the multimeter remote programming command set. This list of commands is specific to his multimeter and is available from the multimeter manufacturer.

He sets up the computer to direct the multimeter to take measurements and record each measurement as it occurs. To do this, he has written an application that uses some simple high-level GPIB commands. The following steps correspond to the program flowchart in Figure 2-1.

1. The application initializes the GPIB by bringing the interface board in the computer online.

2. The application sends the multimeter an instruction, setting it up to take voltage measurements in autorange mode.

3. The application sends the multimeter an instruction to take a voltage measurement.

4. The application tells the multimeter to transmit the data it has acquired to the computer.

   The process of requesting a measurement and reading from the multimeter (Steps 3 and 4) is repeated as long as there are readings to be obtained.

5. As a cleanup step before exiting, the application returns the interface board to its original state by taking it offline.

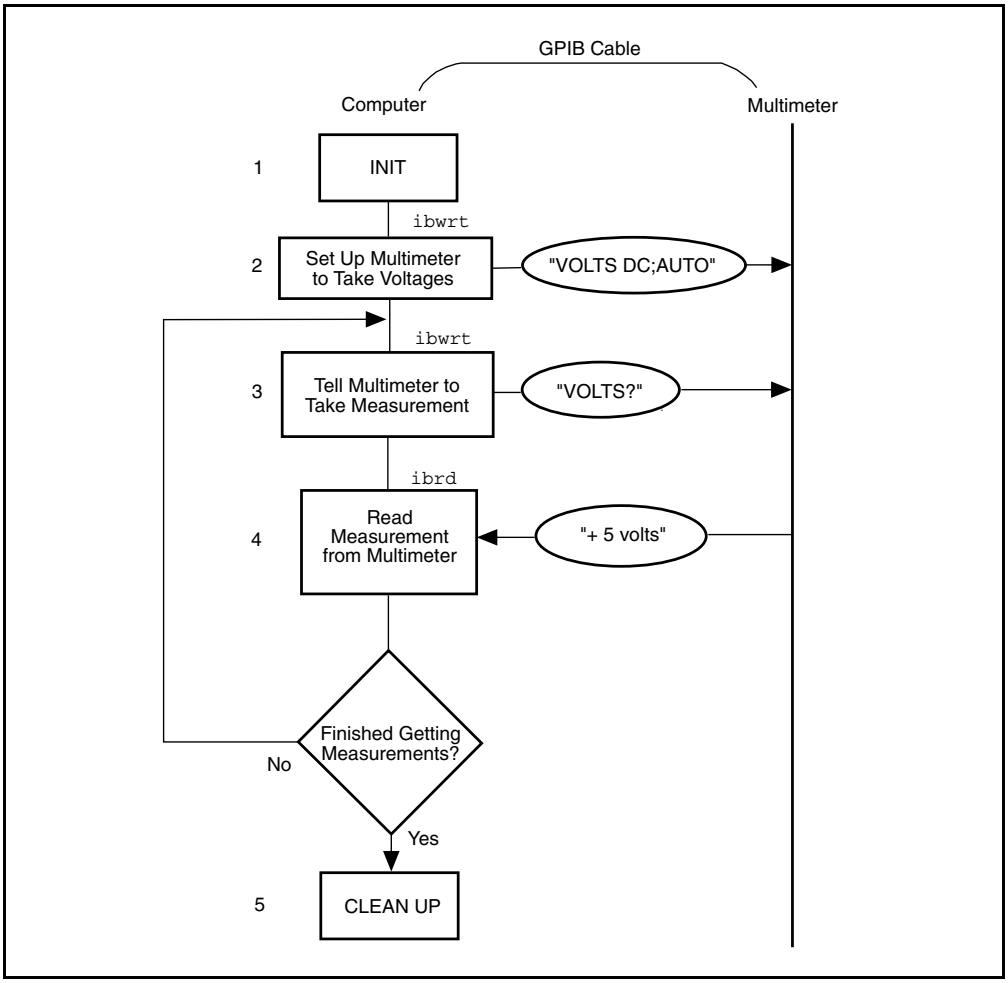


Figure 2-1. Program Flowchart for Example 1

Example 2: Clearing and Triggering Devices

This example illustrates how you can clear and trigger GPIB devices.

Two freshman physics lab partners are learning how to use a GPIB digital oscilloscope. They have successfully loaded the NI-488.2 software on a personal computer and connected their GPIB board to a GPIB digital oscilloscope. Their current lab assignment is to write a small application to practice using the oscilloscope and its command set using high-level GPIB commands. The following steps correspond to the program flowchart in Figure 2-2.

1. The application initializes the GPIB by bringing the interface board in the computer online.

2. The application sends a GPIB clear command to the oscilloscope. This command clears the internal registers of the oscilloscope, reinitializing it to default values and settings.

3. The application sends a command to the oscilloscope telling it to read a waveform each time it is triggered. Predefining the task in this way decreases the execution time required. Each trigger of the oscilloscope is now sufficient to get a new run.

4. The application sends a GPIB trigger command to the oscilloscope which causes it to acquire data.

5. The application queries the oscilloscope for the acquired data. The oscilloscope sends the data.

6. The application reads the data from the oscilloscope.

7. The application calls an external graphics routine to display the acquired waveform.

   Steps 4, 5, 6, and 7 are repeated until all of the desired data has been acquired by the oscilloscope and received by the computer.

8. As a cleanup step before exiting, the application returns the interface board to its original state by taking it offline.

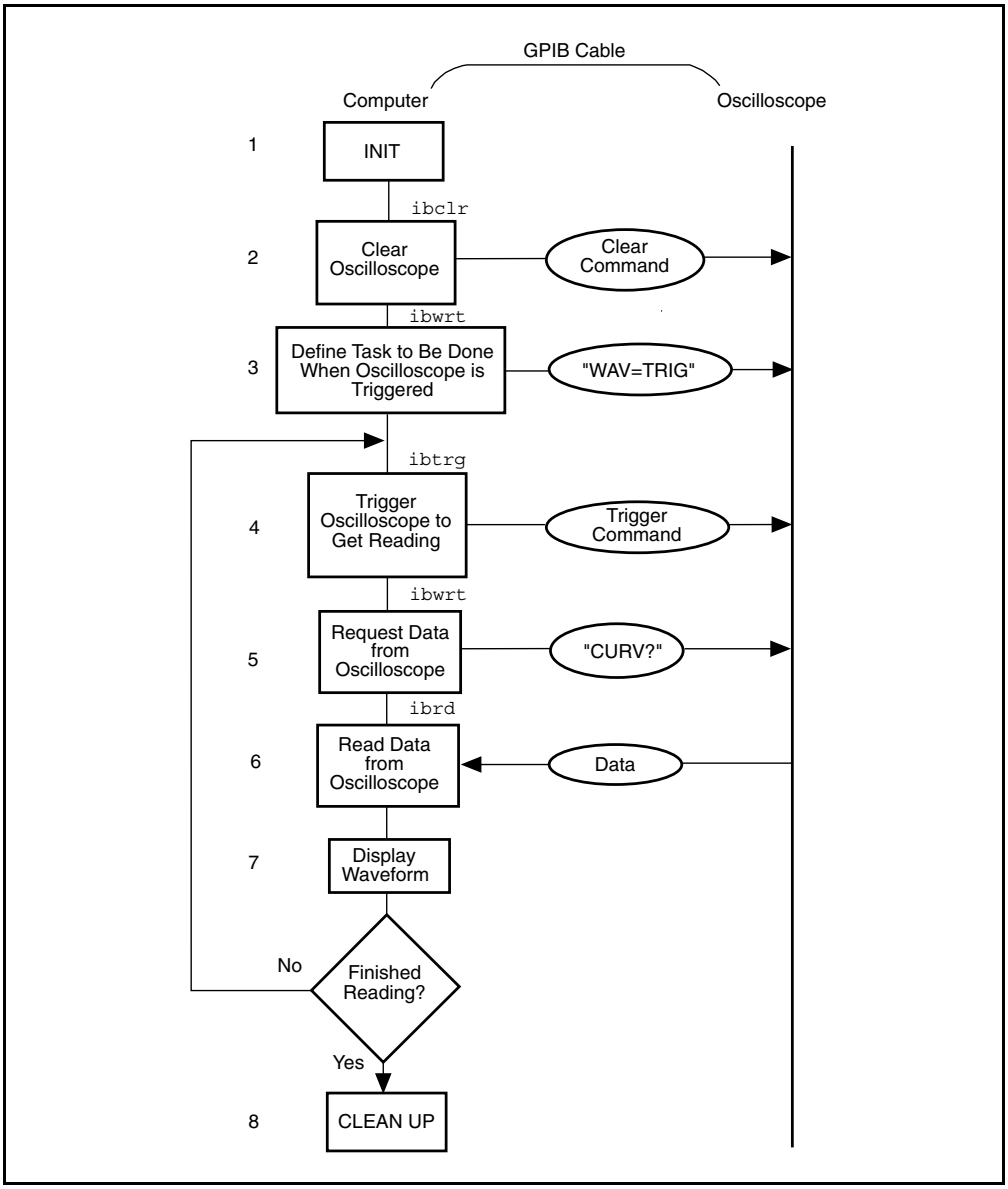


Figure 2-2. Program Flowchart for Example 2

Example 3: Asynchronous I/O

This example illustrates how an application conducts data transfers with a GPIB device and immediately returns to perform other non-GPIB related tasks while GPIB I/O is occurring in the background. This asynchronous mode of operation is particularly useful when the requested GPIB activity may take some time to complete.

In this example, a research biologist is trying to obtain CAT scans of a laboratory animal's liver. She will print out a color copy of each scan as it is acquired. The entire operation is computer-controlled. The CAT scan machine sends the images it acquires to a computer that has the NI-488.2 software installed and is connected to a GPIB color printer . The biologist is familiar with the command set of her color printer, as described in the color printer's user manual. She acquires and prints images with the aid of an application program she wrote using high-level GPIB commands. The following steps correspond to the program flowchart in Figure 2-3.

1. The application initializes the GPIB by bringing the interface board in the computer online.

2. An image is scanned in.

3. The application sends the GPIB printer a command to print the new image and immediately returns without waiting for the I/O operation to be completed.

4. The application saves the image obtained to a file.

5. The application inquires as to whether the printing operation has completed by issuing a GPIB wait command. If the status reported by the wait command indicates completion (CMPL is in the status returned) and more scans need to be acquired, Steps 2 through 5 are repeated until the scans have all been acquired. If the status reported by the wait command in Step 5 does not indicate that printing is finished, statistical computations are performed on the scan obtained and Step 5 is repeated.

6. As a cleanup step before exiting, the application returns the interface board to its original state by taking it offline.

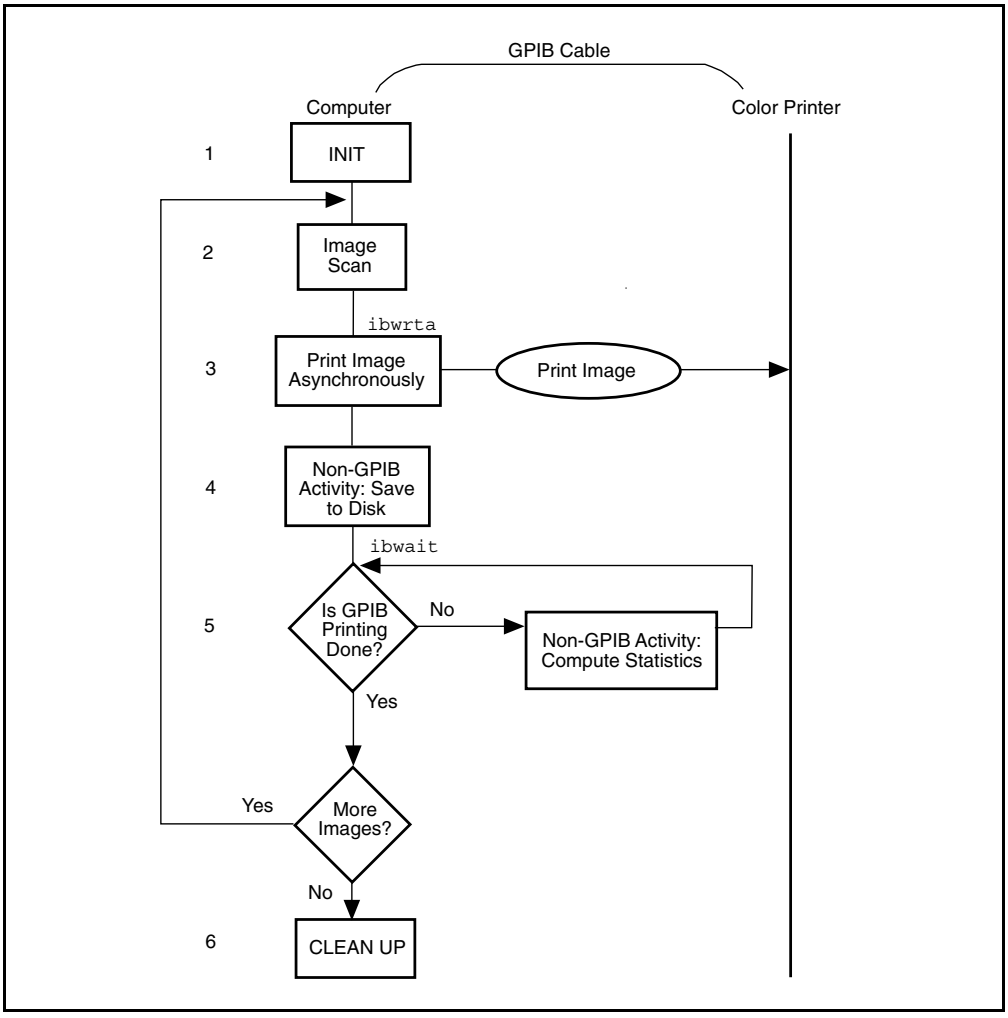


Figure 2-3. Program Flowchart for Example 3

Example 4: End-of-String Mode

This example illustrates how to use the end-of-string modes to detect that the GPIB device has finished sending data.

A journalist is using a GPIB scanner to scan some pictures into his personal computer for a news story. A GPIB cable runs between the scanner and the computer. He is using an application written by an intern in the department who has read the scanner's instruction manual and is familiar with the scanner's programming requirements. The following steps correspond to the program flowchart in Figure 2-4.

1. The application initializes the GPIB by bringing the interface board in the computer online.

2. The application sends a GPIB clear message to the scanner, initializing it to its power-on defaults.

3. The scanner needs to detect a delimiter indicating the end of a command. In this case, the scanner expects the commands to be terminated with <CR><LF> (carriage return, `\r`, and linefeed, `\n`). The application sets its end-of-string (EOS) byte to <LF>. The linefeed code indicates to the scanner that no more data is coming, and is called the end-of-string byte. It flags an end-of-string condition for this particular GPIB scanner. The same effect could be accomplished by asserting the EOI line when the command is sent.

4. With the exception of the scan resolution, all the default settings are appropriate for the task at hand. The application changes the scan resolution by writing the appropriate command to the scanner.

5. The scanner sends back information describing the status of the *change resolution* command. This is a string of bytes terminated by the end-of-string character to tell the application it is done changing the resolution.

6. The application starts the scan by writing the scan command to the scanner.

7. The application reads the scan data into the computer.

8. As a cleanup step before exiting, the application returns the interface board to its original state by taking it offline.

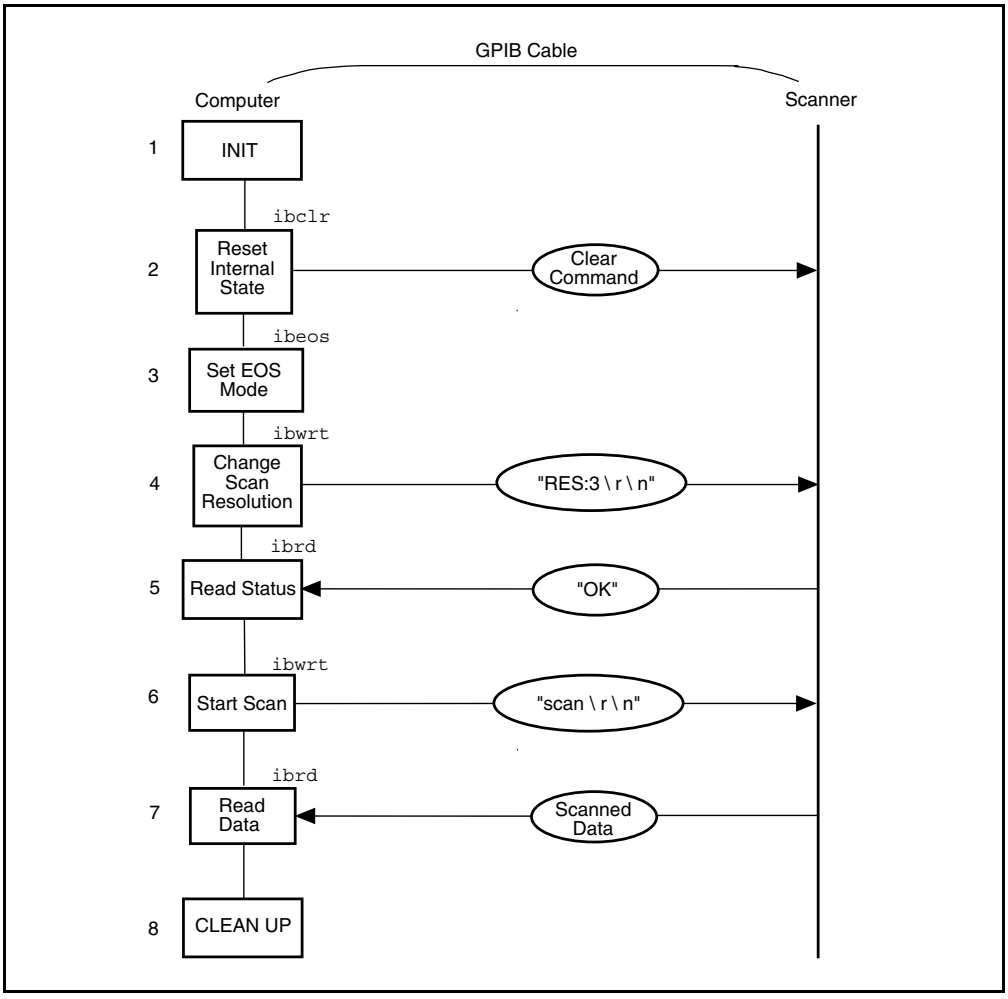


Figure 2-4. Program Flowchart for Example 4

Example 5: Service Requests

This example illustrates how an application communicates with a GPIB device that uses the GPIB service request (SRQ) line to indicate that it needs attention.

A graphic arts designer is transferring digital images stored on her computer to a roll of color film, using a GPIB digital film recorder. A GPIB cable connects the GPIB port on the film recorder to the IEEE 488.2 interface board installed in her computer. She has installed the NI-488.2 software on the host computer and is familiar with the programming instructions for the film recorder, as described in the film recorder's user manual. She places a fresh roll of film in the camera and launches a simple application she has written using high-level GPIB commands. With the aid of the application, she records a few images on film. The following steps correspond to the program flowchart in Figure 2-5.

1. The application initializes the GPIB by bringing the interface board in the computer online.

2. The application brings the film recorder to a ready state by issuing a device clear instruction. The film recorder is now set up for operation using its default values. (The graphic arts designer has previously established that the default values for the film recorder are appropriate for the type of film she is using.)

3. The application advances the new roll of film into position so the first image can be exposed on the first frame of film. This is done by sending the appropriate instructions as described in the film recorder programming guide.

4. The application waits for the film recorder to signify that it is done loading the film, by waiting for RQS (request for service). The film recorder asserts the GPIB SRQ line when it has finished loading the film.

5. As soon as the film recorder asserts the GPIB SRQ line, the application's wait for the RQS event completes. The application conducts a serial poll by sending a special command message to the film recorder that directs it to return a response in the form of a serial poll status byte. This byte contains information indicating what kind of service the film recorder is requesting or what condition it is flagging. In this example, it indicates the completion of a command.

6. A color image transfers to the digital film recorder in three consecutive passes–one pass each for the red, green and blue components of the image. Sub-steps 6a, 6b, and 6c are repeated for each of the passes:

    6a. The application sends a command to the film recorder directing it to accept data to create a single pass image. The film recorder asserts the SRQ line as soon as a pass is completed.

    6b. The application waits for RQS.

6c. When the SRQ line is asserted, the application serial polls the film recorder to see if it requested service, as in Step 5.

7. The application issues a command to the film recorder to advance the film by one frame. The advance occurs successfully unless the end of film is reached.

8. The application waits for RQS, which completes when the film recorder asserts the SRQ line to signal it is done advancing the film.

9. As soon as the application's wait for RQS completes, the application serial polls the film recorder to see if it requested service, as in Step 5. The returned serial poll status byte indicates either of two conditions–the film recorder finished advancing the film as requested or the end of film was reached and it can no longer advance. Steps 6 through 9 are repeated as long as film is in the camera and more images need to be recorded.

10. As a cleanup step before exiting, the application returns the interface board to its original state by taking it offline.

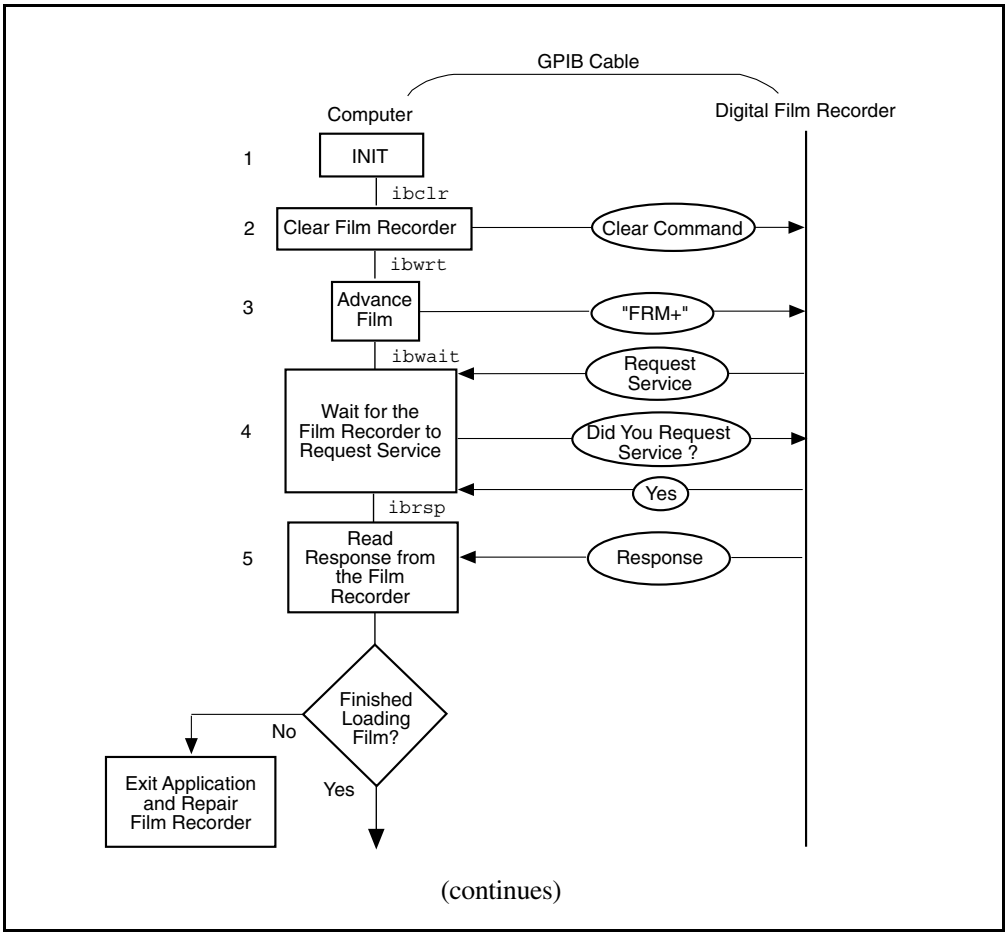


Figure 2-5. Program Flowchart for Example 5

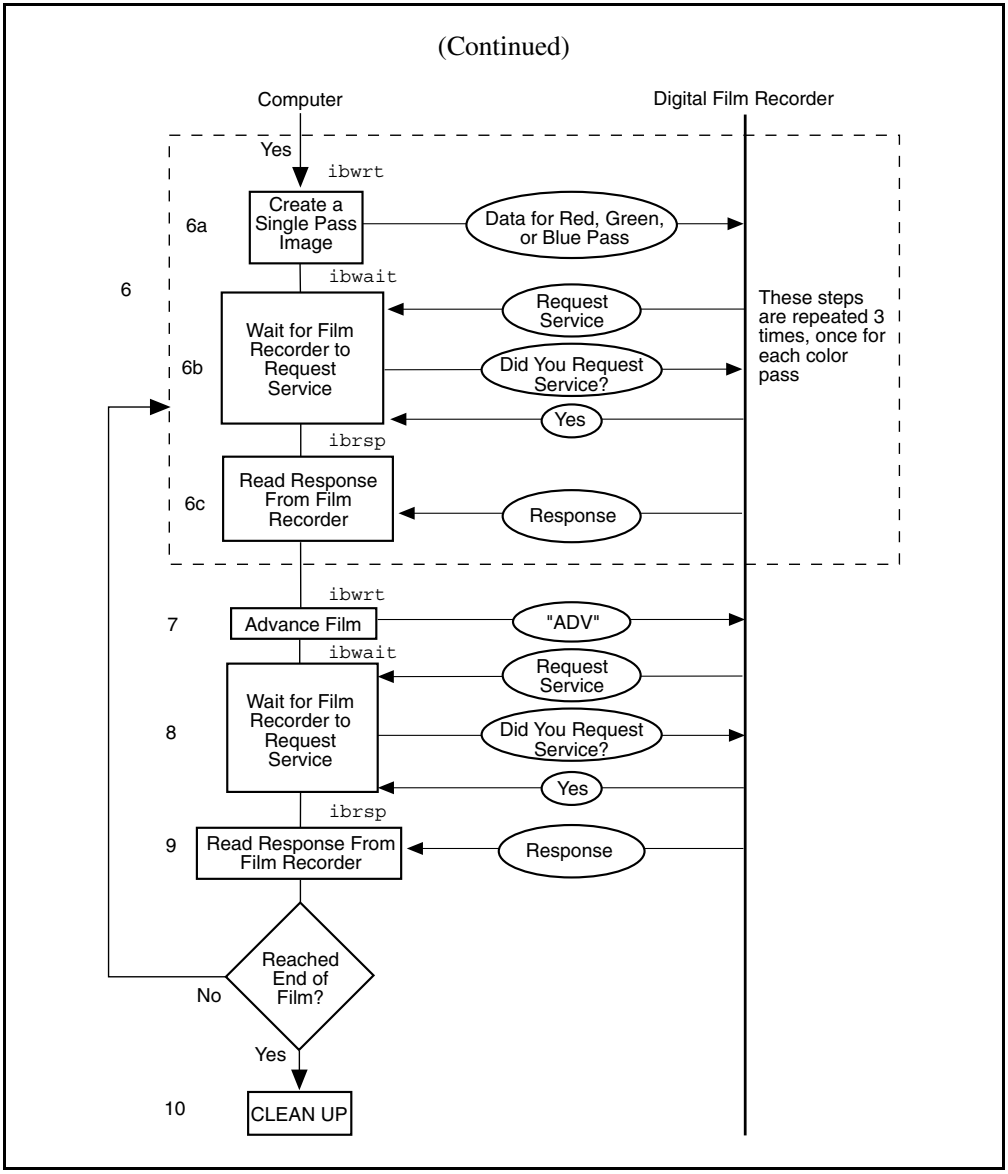


Figure 2-5. Program Flowchart for Example 5 (Continued)

Example 6: Basic Communication with IEEE 488.2-Compliant Devices

This example is an introduction to communicating with IEEE 488.2-compliant devices.

A test engineer in a metal factory is using IEEE 488.2-compliant tensile testers to determine the strength of metal rods as they come out of production. There are several tensile testers and they are all connected to a central computer equipped with an IEEE 488.2 interface board. These machines are fairly voluminous and it is difficult for the engineer to reach the address switches of each machine. For the purposes of his future work with these tensile testers, he needs to determine the setting of each GPIB address switch. He can do so with the aid of a simple application he has written. The following steps correspond to the program flowchart in Figure 2-6.

1. The application initializes the GPIB by bringing the interface board in the computer online.

2. The application issues a command to detect the presence of listening devices on the GPIB and compiles a list of the addresses of all such devices.

3. The application sends an identification query (`"*IDN?"`) to all of the devices detected on the GPIB in Step 2.

4. The application reads the identification information returned by each of the devices as it responds to the query in Step 3.

5. As a cleanup step before exiting, the application returns the interface board back to its original state by taking it offline.

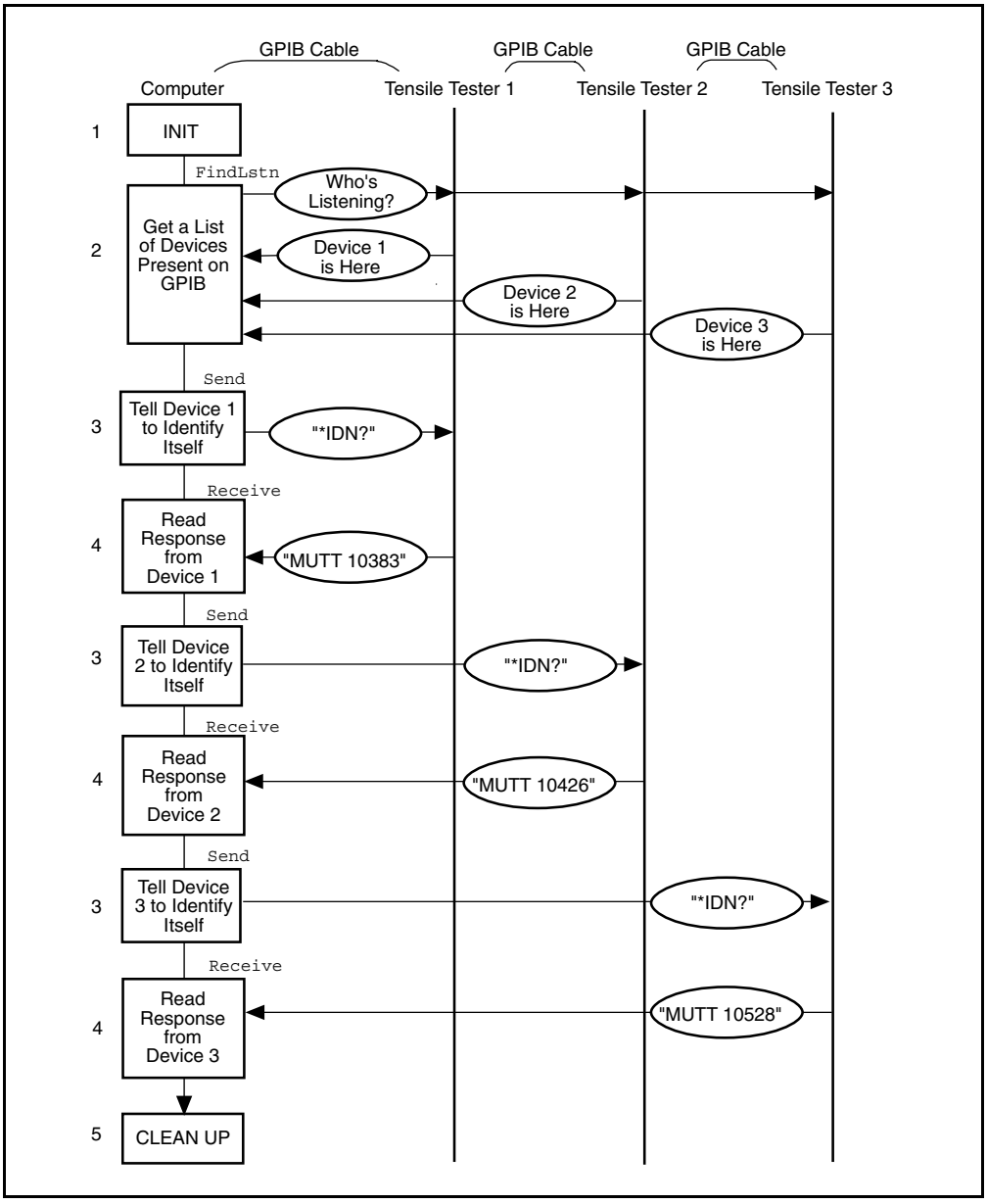


Figure 2-6. Program Flowchart for Example 6

Example 7: Serial Polls Using NI-488.2 Routines

This example illustrates how you can take advantage of the NI-488.2 routines to reduce the complexity of performing serial polls of multiple devices.

A candy manufacturer is using GPIB strain gauges to measure the consistency of the syrup used to make candy. The plant has four big mixers containing syrup. The syrup has to reach a certain consistency to make good quality candy. This is measured by strain gauges that monitor the amount of pressure used to move the mixer arms. When a certain consistency is reached, the mixture is removed and a new batch of syrup is poured in the mixer. The GPIB strain gauges are connected to a computer with an IEEE 488.2 interface board and the NI-488.2 software installed. The process is controlled by an application that uses NI-488.2 routines to communicate with the IEEE 488.2-compliant strain gauges. The following steps correspond to the program flowchart in Figure 2-7.

1. The application initializes the GPIB by bringing the interface board in the computer online.

2. The application configures the strain gauges to request service when they have a significant pressure reading or a mechanical failure occurs. They signal their request for service by asserting the SRQ line.

3. The application waits for one or more of the strain gauges to indicate that they have a significant pressure reading. This wait event ends as soon as the SRQ line is asserted.

4. The application serial polls each of the strain gauges to see if it requested service.

5. Once the application has determined which one of the strain gauges requires service, it takes a reading from that strain gauge.

6. If the reading matches the desired consistency, a dialog window appears on the computer screen and prompts the mixer operator to remove the mixture and start a new batch. Otherwise, a dialog window prompts the operator to service the mixer in some other way.

   Steps 3 through 6 are repeated as long as the mixers are in operation.

7. After the last batch of syrup has been processed, the application returns the interface board to its original state by taking it offline.

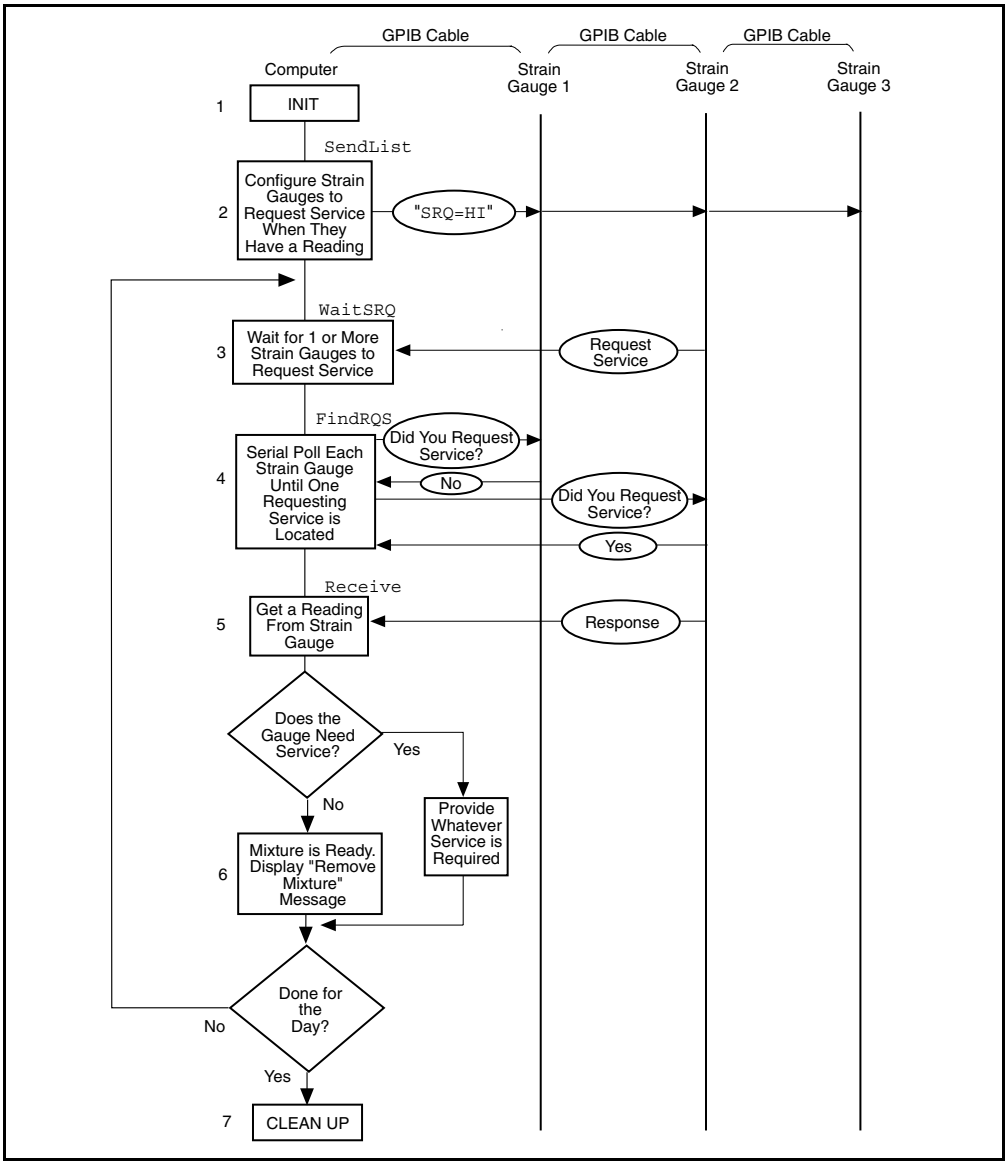


Figure 2-7. Program Flowchart for Example 7

Example 8: Parallel Polls

This example illustrates how you can use NI-488.2 routines to obtain information from several IEEE 488.2-compliant devices at once using a procedure called parallel polling.

The process of manufacturing a particular alloy involves bringing three different metals to specific temperatures before mixing them to form the alloy. Three vats are used, each containing a different metal. Each is monitored by a GPIB ore monitoring unit. The monitoring unit consists of a GPIB temperature transducer and a GPIB power supply. The temperature transducer is used to probe the temperature of each metal. The power supply is used to start a motor to pour the metal into the mold when it reaches a predefined temperature. The three monitoring units are connected to the IEEE 488.2 interface board of a computer that has the NI-488.2 software installed. An application using NI-488.2 routines operates the three monitoring units. The application will obtain information from the multiple units by conducting a parallel poll, and will then determine when to pour the metals into the mixture tank. The following steps correspond to the program flowchart in Figure 2-8.

1. The application initializes the GPIB by bringing the interface board in the computer online.

2. The application configures the temperature transducer in the first monitoring unit by choosing which of the eight GPIB data lines the transducer uses to respond when a parallel poll is conducted. The application also sets the temperature threshold. The transducer manufacturer has defined the individual status (`ist`) bit to be true when the temperature threshold is reached, and the configured status mode of the transducer is *assert the data line*. When a parallel poll is conducted, the transducer asserts its data line if the temperature has exceeded the threshold.

3. The application configures the temperature transducer in the second monitoring unit for parallel polls.

4. The application configures the temperature transducer in the third monitoring unit for parallel polls.

5. The application conducts non-GPIB activity while the metals are heated.

6. The application conducts a parallel poll of all three temperature transducers to determine whether the metals have reached the appropriate temperature. Each transducer asserts its data line during the configuration step if its temperature threshold has been reached.

7. If the response to the poll indicates that all three metals are at the appropriate temperature, the application sends a command to each of the three power supplies, directing them to power on. Then the motors start and the metals pour into the mold.

   If only one or two of the metals is at the appropriate temperature, Steps 5 and 6 are repeated until the metals can be successfully mixed.

8. The application unconfigures all of the transducers so that they no longer participate in parallel polls.

9. As a cleanup step before exiting, the application returns the interface board to its original state by taking it offline.

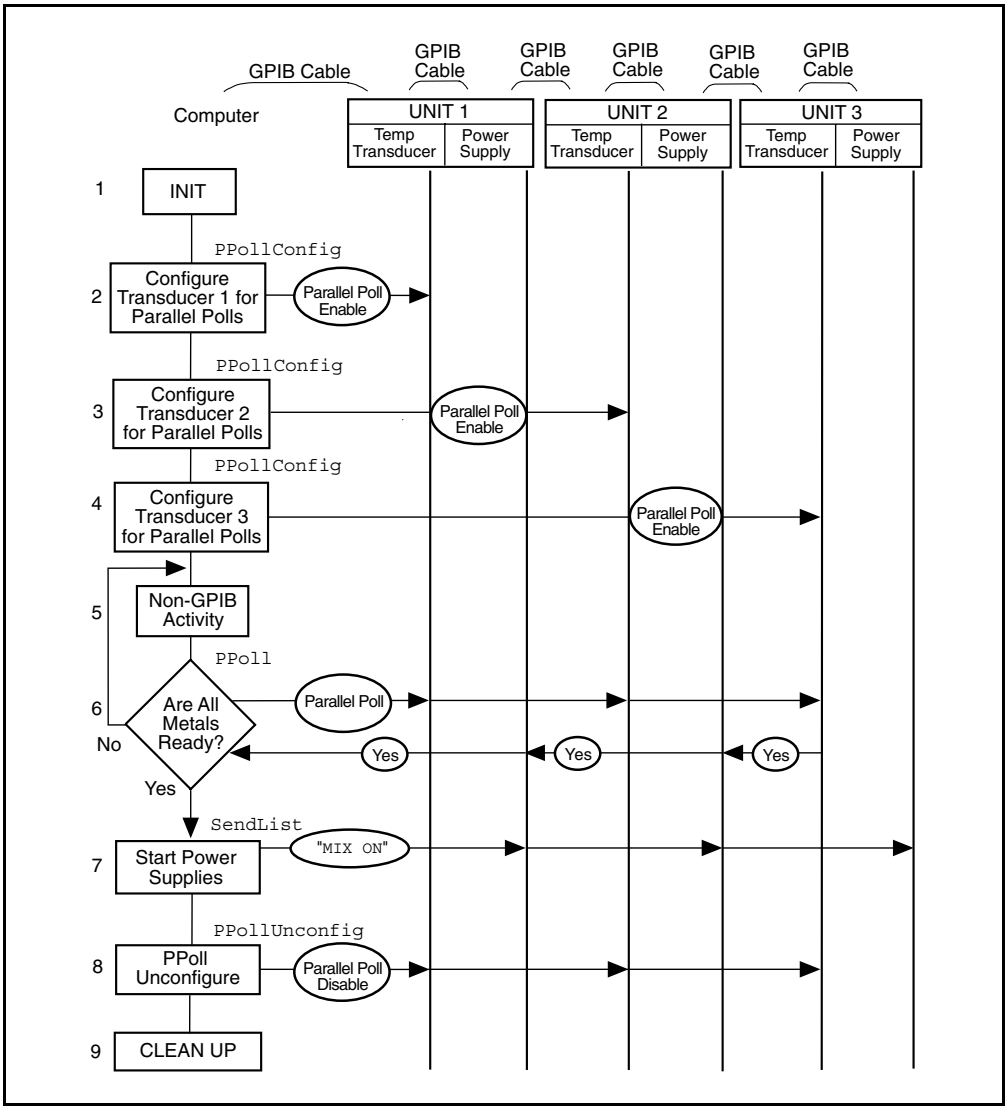


Figure 2-8. Program Flowchart for Example 8

Example 9: Non-Controller Example

This example illustrates how you can use the NI-488.2 software to emulate a GPIB device that is not the GPIB Controller.

A software engineer has written firmware to emulate a GPIB device for a research project and is testing it using an application that makes simple GPIB calls. The following steps correspond to the program flowchart in Figure 2-9.

1. The application brings the device online.

2. The application waits for any of three events to occur: the device to become listen-addressed, become talk-addressed, or receive a GPIB clear message.

3. As soon as one of the events occurs, the application takes an action based upon the event that occurred. If the device was cleared, the application resets the internal state of the device to default values. If the device was talk-addressed, it writes data back to the Controller. If the device was listen-addressed, it reads in new data from the Controller.

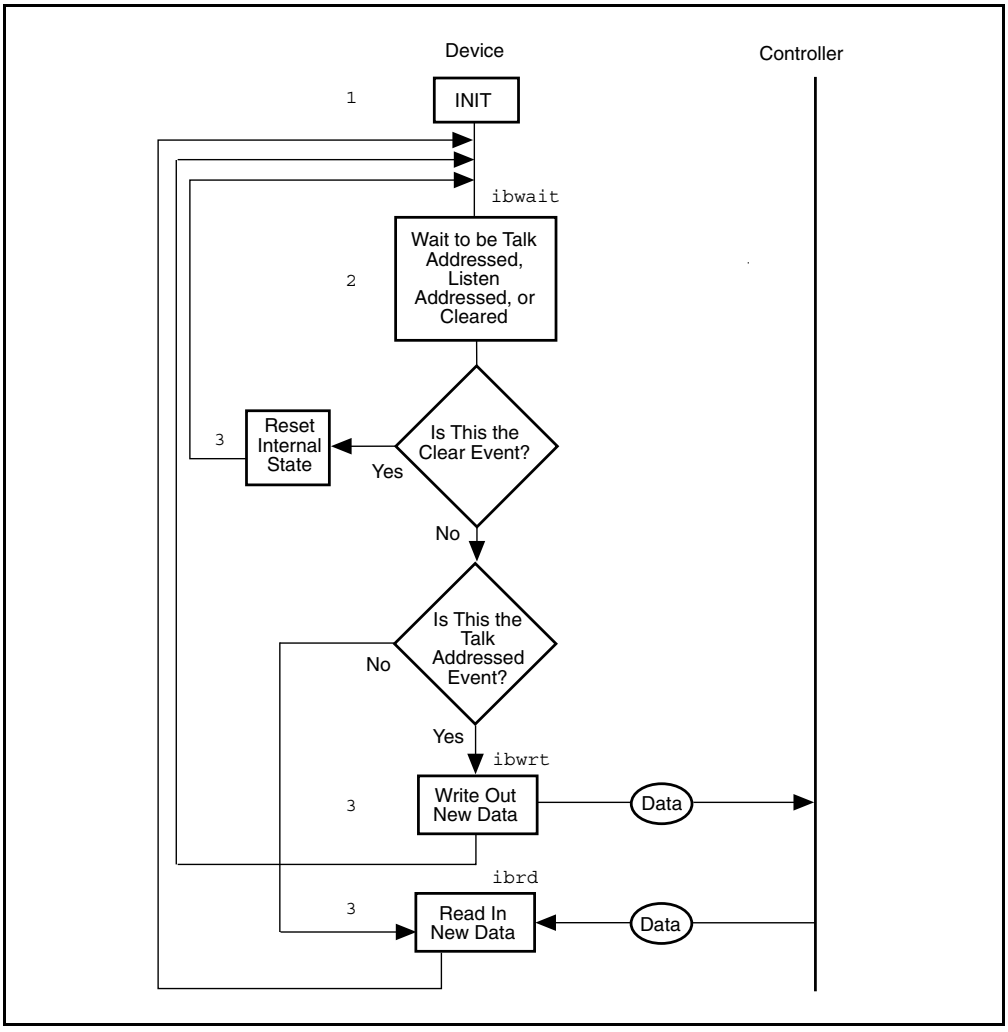


Figure 2-9. Program Flowchart for Example 9

Chapter 3
Developing Your Application

This chapter explains how to develop a GPIB application program using NI-488 functions and NI-488.2 routines.

Choosing How to Access the NI-488.2 DLL

Applications can access the NI-488.2 dynamic link library (`gpib.dll`) either by using the NI-488.2 language interfaces for C or Visual Basic, or by directly accessing the `gpib.dll`. The NI-488.2 language interface is the easiest and fastest method, and it is adequate for most applications.

If you want to access the `gpib.dll` from languages other than C and Visual Basic, or if you have special needs for accessing the `gpib.dll`, then you must use the direct entry functions. These functions are identical to the NI-488 and NI-488.2 calls, except for the following two differences:

- You must preface each function name by the characters `DLL`. For example, instead of using the NI-488 function `ibwrt`, use the direct entry function `DLLibwrt`.
- The number of arguments taken by each function is increased by three. The last three parameters are, in order, a pointer to the `ibsta` variable, a pointer to the `iberr` variable, and a pointer to the `ibcntl` variable. Your application must declare its own `ibsta`, `iberr`, and `ibcntl` variables to use the direct entry functions.

Choosing Between NI-488 Functions and NI-488.2 Routines

Your NI-488.2 software includes two distinct sets of subroutines to meet your application needs. For most application programs, the NI-488 functions are sufficient. You should use the NI-488.2 routines if you have a complex configuration with one or more interface boards and multiple devices.

The following sections discuss some differences between NI-488 functions and NI-488.2 routines.

Using NI-488 Functions: One Device for Each Board

If your system has only one device attached to each board, the NI-488 functions are probably sufficient for your programming needs. Some other factors that make the NI-488 functions more convenient include the following:

- With NI-488 asynchronous I/O functions (`ibcmda`, `ibrda`, and `ibwrta`), you can initiate an I/O sequence while maintaining control over the CPU for non-GPIB tasks.
- NI-488 functions include built-in file transfer functions (`ibrdf` and `ibwrtf`).
- With NI-488 functions, you can control the bus in non-typical ways or communicate with non-compliant devices.

The NI-488 functions consist of high-level (or device) functions that hide much of the GPIB management operations and low-level (or board) functions that offer you more control over the GPIB than NI-488.2 routines. The following sections describe these different function types.

NI-488 Device Functions

Device functions are high-level functions that automatically execute commands that handle bus management operations such as reading from and writing to devices or polling them for status. If you use device functions, you do not need to understand GPIB protocol or bus management. For information about device-level calls and how they manage the GPIB, refer to *Device-Level Calls and Bus Management*, in Chapter 7, *GPIB Programming Techniques.*

NI-488 Board Functions

Board functions are low-level functions that perform rudimentary GPIB operations. Board functions access the interface board directly and require you to handle the addressing and bus management protocol. In cases when the high-level device functions might not meet your needs, low-level board functions give you the flexibility and control to handle situations such as the following:

- Communicating with non-compliant (non-IEEE 488.2) devices
- Altering various low-level board configurations
- Managing the bus in non-typical ways

The NI-488 board functions are compatible with, and can be interspersed within, sequences of NI-488.2 routines. When you use board functions within a sequence of NI-488.2 routines, you do not need a prior call to `ibfind` to obtain a board descriptor. You simply substitute the board index as the first parameter of the board function call. With this flexibility, you can handle non-standard or unusual situations that you cannot resolve using NI-488.2 routines only.

Using NI-488.2 Routines: Multiple Boards and/or Multiple Devices

When your system includes a board that must access more than one device, use the NI-488.2 routines. NI-488.2 routines can perform the following tasks with a single call:

- Find all of the Listeners on the bus
- Find a device requesting service
- Determine the state of the SRQ line, or wait for SRQ to be asserted
- Address multiple devices to listen

Checking Status with Global Variables

Each NI-488 function and NI-488.2 routine updates the global variables to reflect the status of the device or board that you are using. The status word (`ibsta`), the error variable (`iberr`), and the count variables (`ibcnt` and `ibcntl`) contain useful information about the performance of your application program. Your program should check these variables frequently. The following sections describe each of these global variables and how you can use them in your application program.

Status Word – ibsta

All functions update a global status word, `ibsta`, which contains information about the state of the GPIB and the GPIB hardware. Most of the NI-488 functions return the value stored in `ibsta`. You can test for conditions reported in `ibsta` to make decisions about continued processing, or you can debug your program by checking `ibsta` after each call.

`ibsta` is a 16-bit value. A bit value of one (1) indicates that a certain condition is in effect. A bit value of zero (0) indicates that the condition is not in effect. Each bit in `ibsta` can be set for NI-488 device calls (dev), NI-488 board calls and NI-488.2 calls (brd), or all (dev, brd).

Table 3-1 shows the condition that each bit position represents, the bit mnemonics, and the type of calls for which each bit can be set. For a detailed explanation of each of the status conditions, refer to Appendix A, *Status Word Conditions*.

Table 3-1. Status Word (ibsta) Layout

Mnemonic	Bit Pos.	Hex Value	Type	Description
ERR	15	8000	dev, brd	GPIB error
TIMO	14	4000	dev, brd	Time limit exceeded
END	13	2000	dev, brd	END or EOS detected
SRQI	12	1000	brd	SRQ interrupt received
RQS	11	800	dev	Device requesting service
SPOLL	10	400	brd	Board has been serial polled by Controller
EVENT	9	200	brd	DCAS, DTAS, or IFC event has occurred
CMPL	8	100	dev, brd	I/O completed
LOK	7	80	brd	Lockout State
REM	6	40	brd	Remote State
CIC	5	20	brd	Controller-In-Charge
ATN	4	10	brd	Attention is asserted
TACS	3	8	brd	Talker
LACS	2	4	brd	Listener
DTAS	1	2	brd	Device Trigger State
DCAS	0	1	brd	Device Clear State

The language header files included on your distribution disk contain the mnemonic constants for `ibsta`. You can check a bit position in `ibsta` by using its numeric value or its mnemonic constant. For example, bit position 15 (hex 8000) detects a GPIB error. The mnemonic for this bit is ERR. To check for a GPIB error, use either of the following statements after each NI-488 function and NI-488.2 routine as shown:

```
if (ibsta & ERR) gpiberr();
```

or

```
if (ibsta & 0x8000) gpiberr();
```

where `gpiberr()` is an error handling routine.

Error Variable – iberr

If the ERR bit is set in the status word (`ibsta`), a GPIB error has occurred. When an error occurs, the error type is specified by the value in `iberr`.

Note: ***The value in*** `iberr` ***is meaningful as an error type only when the ERR bit is set, indicating that an error has occurred.***

For more information on error codes and solutions refer to Chapter 4, *Debugging Your Application*, or Appendix B, *Error Codes and Solutions.*

Count Variables – ibcnt and ibcntl

The count variables are updated after each read, write, or command function. `ibcnt` is a 16-bit integer and `ibcntl` is a 32-bit integer. If you are reading data, the count variables indicate the number of bytes read. If you are sending data or commands, the count variables reflect the number of bytes sent.

In your application program, you can use the count variables to null-terminate an ASCII string of data received from an instrument. For example, if data is received in an array of characters, you can use `ibcntl` to null-terminate the array and print the measurement on the screen as follows:

```
char rdbuf[512];
ibrd (ud, rdbuf, 20L);
if (!(ibsta & ERR)){
    rdbuf[ibcntl] = '\0';
    printf ("Read:  %s\n", rdbuf);
}
else {
    error();
}
```

`ibcntl` is the number of bytes received. Data begins in the array at index zero (0); therefore, `ibcntl` is the position for the null character that marks the end of the string.

Using wibic to Communicate with Devices

Before you begin writing your application program, you might want to use the `wibic` utility. With `wibic` (Windows Interface Bus Interactive Control), you communicate with your instruments from the keyboard rather than from an application program. You can use `wibic` to learn to communicate with your instruments using the NI-488 functions or NI-488.2 routines. For specific device communication instructions, refer to the user manual that came with your instrument. For information about using `wibic` and for detailed examples, refer to Chapter 5, *wibic–Windows Interface Bus Interactive Control Utility.*

After you have learned how to communicate with your devices in `wibic`, you are ready to begin writing your application program.

Writing Your NI-488 Application

This section discusses items you should include in your application program, general program steps, and an NI-488 example. In this manual the example code is presented in C. The NI-488.2 software includes the source code for C (`devsamp.c`), Visual Basic (`devsamp.bas`), direct entry functions in C (`dlldev.c`), and direct entry functions in Visual Basic (`dlldev.bas`).

The NI-488.2 software also includes the source code for nine application examples, which are described in Chapter 2, *Application Examples*.

Items to Include

- Include the appropriate GPIB header file. This file contains prototypes for the NI-488 functions and constants that you can use in your application program.
- Check for errors after each NI-488 function call.
- Declare and define a function to handle GPIB errors. This function takes the device offline and closes the application. If the function is declared as:

  ```
  void gpiberr (char * msg); /* function prototype */
  ```

  then your application invokes it as follows:

  ```
  if (ibsta & ERR) {
      gpiberr ("GPIB error");
  }
  ```

NI-488 Program Shell

Figure 3-1 is a flowchart of the steps to create your application program using device-level NI-488 functions.

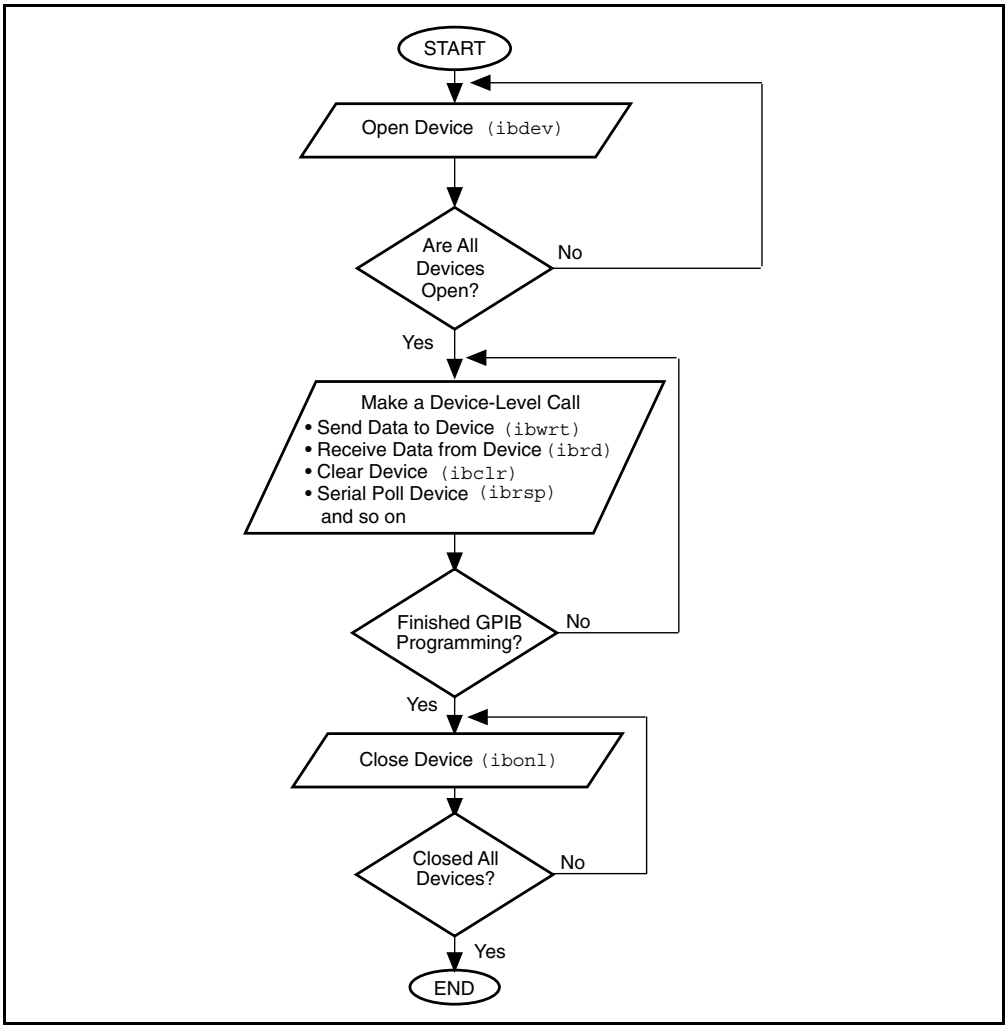


Figure 3-1. General Program Shell Using NI-488 Device Functions

General Program Steps and Examples

The following steps demonstrate how to use the NI-488 device functions in your program. This example configures a digital multimeter, reads 10 voltage measurements, and computes the average of these measurements.

Step 1. Open a Device

Your first NI-488 function call should be to `ibdev` to open a device.

```
ud = ibdev(0, 1, 0 , T10s, 1, 0);

if (ibsta & ERR) {
   gpiberr("ibdev error");
}
```

The input arguments of the `ibdev` function are as follows:

0 - board index for GPIB0

1 - primary GPIB address of the device

0 - no secondary GPIB address for the device

T10s - I/O timeout value (10 s)

1 - send END message with the last byte when writing to device

0 - disable EOS detection mode

When you call `ibdev`, the driver automatically initializes the GPIB by sending an Interface Clear (IFC) message and placing the device in remote programming state.

Step 2. Clear the Device

Clear the device before you configure the device for your application. Clearing the device resets its internal functions to a default state.

```
ibclr(ud);
if (ibsta & ERR) {
   gpiberr("ibclr error");
}
```

Step 3. Configure the Device

After you open and clear the device, it is ready to receive commands. To configure the instrument, you send device-specific commands using the `ibwrt` function. Refer to the instrument user manual for the command bytes that work with your instrument.

```
ibwrt(ud, "*RST; VAC; AUTO; TRIGGER 2; *SRE 16", 35L);
if (ibsta & ERR) {
   gpiberr("ibwrt error");
}
```

The programming instruction in this example resets the multimeter (`*RST`). The meter is instructed to measure the volts alternating current (`VAC`) using auto-ranging (`AUTO`), to wait for a trigger from the GPIB interface board before starting a measurement (`TRIGGER 2`), and to assert the SRQ line when the measurement completes and the multimeter is ready to send the result (`*SRE 16`).

Step 4. Trigger the Device

If you configure the device to wait for a trigger, you must send a trigger command to the device before reading the measurement value. Then instruct the device to send the next triggered reading to its GPIB output buffer.

```
ibtrg(ud);
   if (ibsta & (ERR 1 TIMO)) {
   gpiberr("ibtrg error");
}

ibwrt(ud,"VAL1?", 5L);
if (ibsta & ERR) {
   gpiberr("ibwrt error");
}
```

Step 5. Wait for the Measurement

After you trigger the device, the RQS bit is set when the device is ready to send the measurement. You can detect RQS by using the `ibwait` function. The second parameter indicates what you are waiting for. Notice that the `ibwait` function also returns when the I/O timeout value is exceeded.

```
printf("Waiting for RQS...\n");
ibwait (ud, TIMO| RQS);
if (ibsta & (ERR | TIMO)) {
   gpiberr("ibwait error");
}
```

When SRQ has been detected, serial poll the instrument to determine if the measured data is valid or if a fault condition exists. For IEEE 488.2 instruments, you can find out by checking the message available (MAV) bit, bit 4 in the status byte that you receive from the instrument.

```
ibrsp (ud, &StatusByte);
if (ibsta & ERR) {
   gpiberr("ibrsp error");
}

if ( !(StatusByte & MAVbit)) {
   gpiberr("Improper Status Byte");
   printf("  Status Byte = 0x%x\n", StatusByte);
}
```

Step 6. Read the Measurement

If the data is valid, read the measurement from the instrument. (`AsciiToFloat` is a function that takes a null-terminated string as input and outputs the floating point number it represents.)

```
ibrd (ud, rdbuf, 10L);
if (ibsta & ERR) {
   gpiberr("ibrd error");
}

rdbuf[ibcntl] = '\0';
printf("Read: %s\n", rdbuf);
      /*  Output ==> Read: +10.98E-3  */

sum += AsciiToFloat(rdbuf);
```

Step 7. Process the Data

Repeat steps 4 through 6 in a loop until 10 measurements have been read. Then print the average of the readings as shown:

```
printf("The average of the 10 readings is %f\n", sum/10.0);
```

Step 8. Place the Device Offline

As a final step, take the device offline using the `ibonl` function.

```
ibonl (ud, 0);
```

Writing Your NI-488.2 Application

This section discusses items you should include in an application program that uses NI-488.2 routines, general program steps, and an NI-488.2 example. In this manual the example code is presented in C. The NI-488.2 software includes the source code for C (`samp4882.c`), Visual Basic (`samp4882.bas`), direct entry functions in C (`dll4882.c`), and direct entry functions in Visual Basic (`dll4882.bas`).

The NI-488.2 software also includes the source code for nine example programs, which are described in Chapter 2, *Application Examples*.

Items to Include

- Include the appropriate GPIB header file. This file contains prototypes for the NI-488.2 routines and constants that you can use in your application program.
- Check for errors after each NI-488.2 routine.
- Declare and define a function to handle GPIB errors. This function takes the device offline and closes the application. If the function is declared as:

```
void gpiberr (char * msg); /* function prototype */
```

  then your application invokes it as follows:

```
if (ibsta & ERR) {
    gpiberr ("GPIB error");
}
```

NI-488.2 Program Shell

Figure 3-2 is a flowchart of the steps to create your application program using NI-488.2 routines.

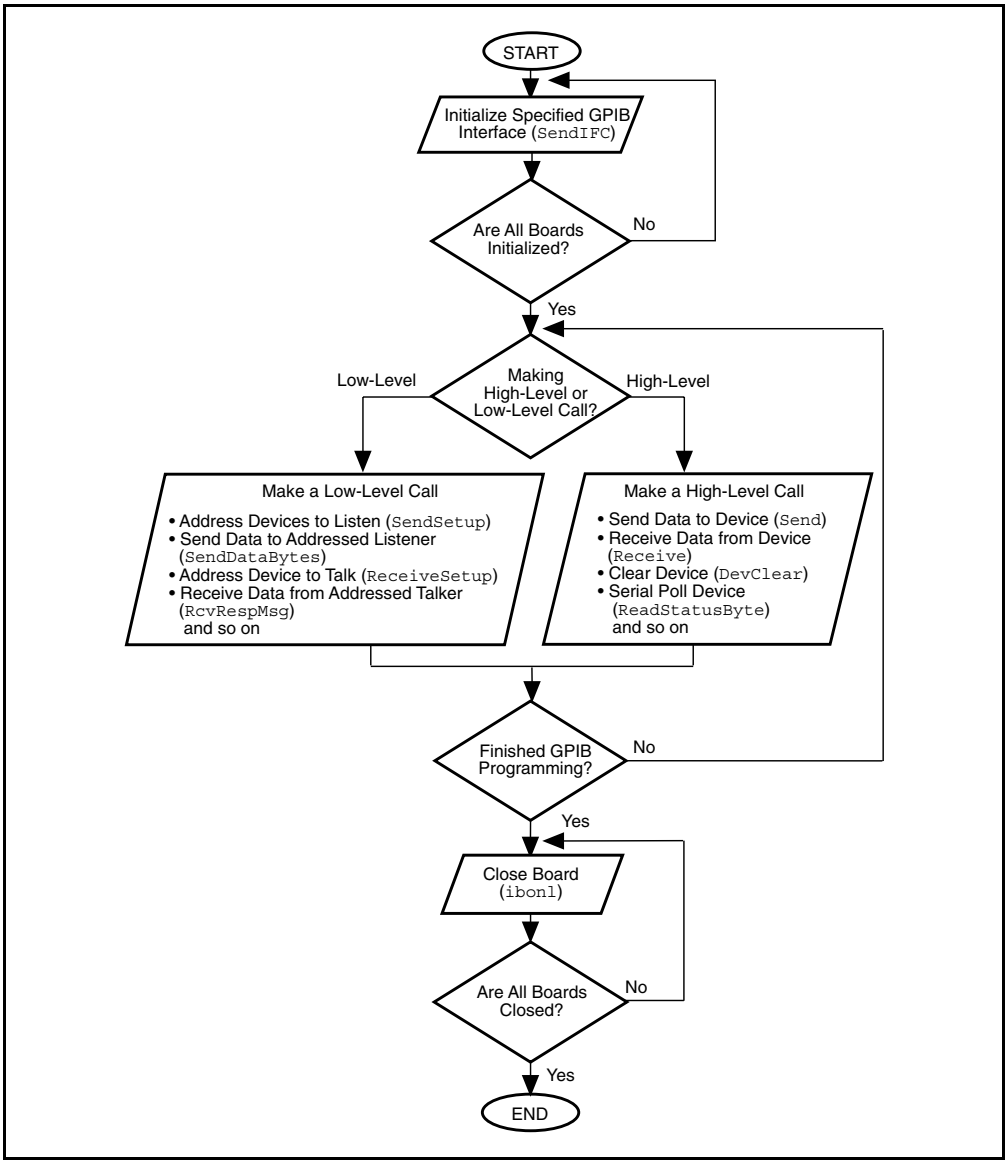


Figure 3-2. General Program Shell Using NI-488.2 Routines

General Program Steps and Examples

The following steps demonstrate how to use the NI-488.2 routines in your program. This example configures a digital multimeter, reads 10 voltage measurements, and computes the average of these measurements.

Step 1. Initialization

Use the `SendIFC` routine to initialize the bus and the GPIB interface board so that the GPIB board is Controller-In-Charge (CIC). The only argument of `SendIFC` is the GPIB interface board number.

```
SendIFC(0);
if (ibsta & ERR) {
   gpiberr("SendIFC error");
}
```

Step 2. Find All Listeners

Use the `FindLstn` routine to create an array of all of the instruments attached to the GPIB. The first argument is the interface board number, the second argument is the list of instruments that was created, the third argument is a list of instrument addresses that the procedure actually found, and the last argument is the maximum number of devices that the procedure can find (that is, it must stop if it reaches the limit). The end of the list of addresses must be marked with the `NOADDR` constant, which is defined in the header file that you included at the beginning of the program.

```
for (loop = 0; loop <=30; loop++){
   instruments[loop] = loop;
}
instruments[31] = NOADDR;

printf("Finding all Listeners on the bus...\n");

Findlstn(0, instruments, result, 30);
if (ibsta & ERR) {
   gpiberr("FindLstn error");
}
```

Step 3. Identify the Instrument

Send an identification query to each device for identification. For this example, assume that all of the instruments are IEEE 488.2-compatible and can accept the identification query, `*IDN?`. In addition, assume that `FindLstn` found the GPIB interface board at primary address 0 (default) and, therefore, you can skip the first entry in the `result` array.

```
for (loop = 1; loop <= num_Listeners; loop++) {
    Send(0, result[loop], "*IDN?", 5L, NLend);
    if  (ibsta & ERR) {
        gpiberr("Send error");
    }

    Receive(0, result[loop], buffer, 10L, STOPend);
    if  (ibsta & ERR) {
        gpiberr("Receive error");
    }

    buffer[ibcntl] = '\0';
    printf("The instrument at address %d is a %s\n",
            result[loop], buffer);
    if (strncmp(buffer, "Fluke, 45", 9) == 0) {
        fluke = result[loop];
        printf("****  Found the Fluke ****\n");
        break;
    }
}

if (loop > num_Listeners) {
    printf("Did not find the Fluke!\n");
    ibonl(0,0);
    exit(1);
}
```

The constant `NLend` signals that the new line character with EOI is automatically appended to the data to be sent.

The constant `STOPend` indicates that the read is stopped when EOI is detected.

Step 4. Initialize the Instrument

After you find the multimeter, use the `DevClear` routine to clear it. The first argument is the GPIB board number. The second argument is the GPIB address of the multimeter. Then send the IEEE 488.2 reset command to the meter.

```
DevClear(0, fluke);
if (ibsta & ERR) {
    gpiberr("DevClear error")
}

Send(0, fluke, "*RST", 4L, NLend);
if  (ibsta & ERR) {
    gpiberr("Send *RST error");
}
sum = 0.0;
for(m =0; m<10; m++){
/* start of loop for Steps 5 through 8 */
```

Step 5. Configure the Instrument

After initialization, the instrument is ready to receive instructions. To configure the multimeter, use the `Send` routine to send device-specific commands. The first argument is the number of the access board. The second argument is the GPIB address of the multimeter. The third argument is a string of bytes to send to the multimeter.

The bytes in this example instruct the meter to measure volts alternating current (`VAC`) using auto-ranging (`AUTO`), to wait for a trigger from the Controller before starting a measurement (`TRIGGER 2`), and to assert SRQ when the measurement has been completed and the meter is ready to send the result (`*SRE 16`). The fourth argument represents the number of bytes to be sent. The last argument, `NLend`, is a constant defined in the header file which tells `Send` to append a linefeed character, with EOI asserted, to the end of the message sent to the multimeter.

```
Send (0, fluke, "VAC; AUTO; TRIGGER 2; *SRE 16", 29L, NLend);
if (ibsta & ERR) {
    gpiberr("Send setup error");
}
```

Step 6. Trigger the Instrument

In the previous step, the multimeter was instructed to wait for a trigger before conducting a measurement. Now send a trigger command to the multimeter. You could use the `Trigger` routine to accomplish this, but because the Fluke 45 is IEEE 488.2-compatible, you can just send it the trigger command, `*TRG`. The `VAL1?` command instructs the meter to send the next triggered reading to its output buffer.

```
Send(0, fluke, "*TRG; VAL1?", 11L, NLend);
if (ibsta & ERR) {
    gpiberr("Send trigger error");
}
```

Step 7. Wait for the Measurement

After the meter is triggered, it takes a measurement and displays it on its front panel and then asserts SRQ. You can detect the assertion of SRQ using either the `TestSRQ` or `WaitSRQ` routine. If you have a process that you want to execute while you are waiting for the measurement, use `TestSRQ`. For this example, you can use the `WaitSRQ` routine. The first argument in `WaitSRQ` is the GPIB board number. The second argument is a flag returned by `WaitSRQ` that indicates whether or not SRQ is asserted.

```
WaitSRQ(0, &SRQasserted);
if (!SRQasserted) {
    gpiberr("WaitSRQ error");
}
```

After you have detected SRQ, use the `ReadStatusByte` routine to poll the meter and determine its status. The first argument is the GPIB board number, the second argument

is the GPIB address of the instrument, and the last argument is a variable that `ReadStatusByte` uses to store the status byte of the instrument.

```
ReadStatusByte(0, fluke, &statusByte);
if (ibsta & ERR) {
   gpiberr("ReadStatusByte error");
}
```

After you have obtained the status byte, you must check to see if the meter has a message to send. You can do this by checking the message available (MAV) bit, bit 4 in the status byte.

```
if (!(statusByte & MAVbit) {
   gpiberr("Improper Status Byte");
   printf("Status Byte = 0x%x\n", statusByte);
}
```

Step 8. Read the Measurement

Use the `Receive` function to read the measurement over the GPIB. The first argument is the GPIB interface board number, and the second argument is the GPIB address of the multimeter. The third argument is a string into which the `Receive` function places the data bytes from the multimeter. The fourth argument represents the number of bytes to be received. The last argument indicates that the `Receive` message terminates upon receiving a byte accompanied with the END message.

```
Receive(0, fluke, buffer, 10L, STOPend);
if (ibsta & ERR) {
   gpiberr("Receive error");
}
```

```
buffer[ibcntl] = '\0';
printf (Reading : %s\n", buffer);
sum += AsciiToFloat(buffer);
}  /*  end of loop started in Step 5 */
```

Step 9. Process the Data

Repeat Steps 5 through 8 in a loop until 10 measurements have been read. Then print the average of the readings as shown:

```
printf (" The average of the 10 readings is : %f\n", sum/10);
```

Step 10. Place the Board Offline

Before ending your application program, take the board offline using the `ibonl` function.

```
ibonl(0,0);
```

Compiling, Linking, and Running Your Application

The following sections describe how to compile, link, and run your application program. To see examples that show how to include specific lines of code in your program, refer to the program examples included with your NI-488.2 software.

If you discover errors when you execute the program refer to Chapter 4, *Debugging Your Application*. If you want to verify the NI-488 and NI-488.2 calls made by your application, you can use GPIB Spy, the GPIB applications monitor for Windows described in Chapter 6, *GPIB Spy*.

Microsoft C

Before you compile your C application program, make sure that the following line is included at the beginning of your program:

```
#include "windecl.h"
```

After you have written your C application program, you must compile your application program using one of the following compilers: Microsoft C Version 7.0 with Microsoft SDK (Software Development Kit) Version 3.1 or Microsoft Visual C++ Version 1.0 (also known as Microsoft C Version 8.0). Then link your application with the C language interface, `gpib.lib`. Use the following command to compile and link a QuickWin C application named `cprog` using Microsoft C:

```
cl /Mq cprog.c gpib.lib
```

The `/Mq` option of Microsoft C creates a QuickWin application. To run your application, choose the **Run** option from the **File** menu in the **Program Manager** window. Enter the path and name of the compiled program in the dialog box that pops up.

Borland C++

Before you compile your C application program, make sure that the following line is included at the beginning of your program:

```
#include "windecl.h"
```

After you have written your C application program, you must compile your application program using the Borland C++ (version 2.0 or higher) compiler. Then link your application with the C language interface, `gpib.lib`. Use the following command to compile and link a Windows C application named `cprog` using Borland C++:

```
bcc -W cprog.c gpib.lib
```

The `-W` option of Borland C++ creates a Windows application. To run your application, choose the **Run** option from the **File** menu in the **Program Manager** window. Enter the path and name of the compiled program in the dialog box that pops up.

Visual Basic

With Visual Basic, you access the NI-488 functions as subroutines, using the BASIC keyword CALL followed by the NI-488 function name (for example `CALL ibtrg (ud%)`). You can also access the NI-488 functions using another set of functions, the `il` functions (for example `result% = iltrg (ud%)`). If you are using Visual Basic and prefer calling functions instead of subroutines, then you can use the `il` functions. With some of the `ib` commands (for example, `ibrd`) the length of the string buffer is automatically calculated within the actual subroutine, which eliminates the need to pass in the length as an extra parameter.

Before you run your Visual Basic application, include the files `niglobal.bas` and `vbib.bas` in your application project file.

If you are using Microsoft Visual Basic 1.0, you must edit `niglobal.bas` to define the constants `true` and `false` as follows:

```
global const true  = -1
global const false =  0
```

In Microsoft Visual Basic 2.0 and higher, `true` and `false` are reserved words; therefore, these constants are no longer defined in `niglobal.bas`.

After you have written your Visual Basic application, choose the **Start** option from the **Run** menu to execute your program.

Direct Entry with C

Before you compile your C application program, make sure that the following lines are included at the beginning of your program:

```
#ifdef __cplusplus
extern "C"{
#endif

#include "windecl.h

int ibsta, iberr;          /* NI-488.2 global status variables */
long ibcntl;

#ifdef __cplusplus
}
#endif
```

You must define the global status variables in your application program.

Sample programs using direct entry are included with the National Instruments driver. The C sample programs are console-type applications. They illustrate how to use the function calls the NI-488.2 driver exports. Since most of the popular compilers support console-type applications as well as GUI applications, you can create console-type applications without knowing Windows GUI programming. A console-type application is a Win16 program that uses text-based input and output instead of a graphical interface. This allows you to quickly create a Win16 application by using simple input and output functions like `printf` and `scanf`.

You must define the direct entry prototypes of the functions used in your application program. There are three methods of accessing the Dynamic Link Library (DLL) directly. The *NI-488.2 Function Reference Manual for DOS/Windows* provides the format for the first two methods. Refer to it for the C direct entry syntax for NI-488 functions and NI-488.2 routines. The direct entry sample programs use the third method. The three methods of direct entry are documented in the Microsoft Windows Software Development Kit (SDK) Guide's online help file. The first method of accessing the DLL follows.

Method 1. List Functions in the Module Definition File

To use the first method, list the names of the functions in the `imports` section in the application's `.def` file.

You must create a module definition file, such as `cprog.def`. This file contains an `imports` section that lists the direct entry points used by the application program. You must link the module definition file to your application program.

After you have written your Microsoft C application program, compile your program and use the `/Mq` option of Microsoft C to create a QuickWin application. This option is available starting with Microsoft C/C++ Version 7.0 with the Microsoft Software Development Kit (SDK) or with Microsoft Visual C++ Version 1.*x*. The following command compiles and links a Microsoft QuickWin application named `cprog` with the module definition file `cprog.def`:

```
cl /Mq cprog.c cprog.def
```

After you have written your Borland C/C++ application program, compile your program and use the `-W` option of Borland C/C++ to create a Windows application. The following command compiles and links a Borland C/C++ Windows application named `cprog` with the module definition file `cprog.def`:

```
bcc -W cprog.c cprog.def
```

To run your application, choose the **Run** option from the **File** menu in the **Program Manager** window. Enter the path and name of the compiled program in the dialog box that appears.

The second method of accessing the DLL follows.

Method 2. Generate an Import Library Using implib

To use the second method, link the application to an import library generated from the DLL using the `implib` utility.

To generate the import library, type in the following text:

```
implib gpib.lib c:\windows\system\gpib.dll
```

where `c:\windows\system\` is the path to your Windows System directory. The file named `gpib.lib` is the name of the file generated by the `implib` utility. For more information about the `implib` utility program, refer to the Windows online help or other documentation supplied with your compiler.

After you have written your Microsoft C application program, compile your program and use the `/Mq` option of Microsoft C to create a QuickWin application. This option is available starting with Microsoft C/C++ Version 7.0 with the Microsoft Software Development Kit (SDK) or with Microsoft Visual C++ Version 1.*x*. The following command compiles and links a Microsoft QuickWin application named `cprog` with the import library file `gpib.lib`:

```
cl /Mq cprog.c gpib.lib
```

After you have written your Borland C/C++ application program, compile your program and use the `-W` option of Borland C/C++ to create a Windows application. The following command compiles and links a Borland C/C++ Windows application named `cprog` with the import library file `gpib.lib`:

```
bcc -W cprog.c gpib.lib
```

To run your application, choose the **Run** option from the **File** menu in the **Program Manager** window. Enter the path and name of the compiled program in the dialog box that appears.

The third method of accessing the DLL follows.

Method 3. Use LoadLibrary and GetProcAddress

To use the third method, use the `LoadLibrary` and `GetProcAddress` functions to load the library and retrieve a pointer to the function.

The Direct Entry sample programs use this method. At the beginning of your program, declare the following global variable for the DLL as follows:

```
HINSTANCE GpibLib = NULL;
```

The prototypes for each function can be found in the *NI-488.2 Function Reference Manual for DOS/Windows*. For functions that return an integer value, like `ibdev`, you must cast the pointer to the function as follows:

```
int(_far _pascal *Pname)
```

where `*Pname` is the name of the pointer to the function. For functions (that is, the NI-488.2 calls) that do not return a value, you must cast the pointer to the function as follows:

```
void(_far _pascal *Pname)
```

where `*Pname` is the name of the pointer to the function. A function is followed by its list of parameters and pointers to the three global variables, as described in the *NI-488.2 Function Reference Manual for DOS/Windows*. The following is an example of how to cast the function pointer and how to set up the parameter list for `ibdev` and `ibonl`:

```
int(_far _pascal *Pibdev)(int BdIndx, int pad, int sad,
             int tmo, int eot, int eos, int _far *ibsta,
             int _far *iberr, long _far *ibcntl);
int(_far _pascal *Pibonl)(int ud, int v, int _far *ibsta,
             int _far *iberr, long _far *ibcntl);
```

You must load the `GPIB.DLL` library within your application program. The following code calls the `LoadLibrary` function and handles errors:

```
GpibLib=LoadLibrary("GPIB.DLL");
if (GpibLib <= HINSTANCE_ERROR)
{
return FALSE;
}
```

Next you must get the address and assign it to the function pointer by using the `GetProcAddress` function. This code goes in your application program.

```
Pibdev=(int (_far _pascal *)(int, int, int, int, int, int,
          int _far*, int _far *, long _far *))
          GetProcAddress(GpibLib,(LPCSTR)"DLLibdev");
Pibonl=(int (_far _pascal *)(int, int, int _far *, int _far *,
          long _far *))GetProcAddress(GpibLib,
          (LPCSTR)"DLLibonl");
```

If `GetProcAddress` does not return a pointer, you can check to see if an address was assigned or not, as in the following example:

```
if ((Pibdev==NULL)||
    (Pibonl==NULL))
    {
        FreeLibrary(GpibLib);
        GpibLib=NULL;
        return FALSE;
    }
    else
    return TRUE;
```

The following code shows you how to call functions from within your application program:

```
dvm = (*Pibdev) (0, 1, 0, T10s, 1, 0, &ibsta, &iberr,
                 &ibcntl);
(*Pibonl) (dvm, 0, &ibsta, &iberr, &ibcntl);
```

Upon completion of your program, you must free the library, as in the following example:

```
FreeLibrary(GpibLib);
```

For more information about the Windows functions like `LoadLibrary` and `GetProcAddress`, refer to the Windows SDK documentation or your compiler's Windows documentation.

After you have written your Microsoft C application program, compile your program and use the `/Mq` option of Microsoft C to create a QuickWin application. This option is available starting with Microsoft C/C++ Version 7.0 with the Microsoft Software Development Kit (SDK) or with Microsoft Visual C++ Version 1.x. The following command compiles and links a Microsoft QuickWin application named `cprog`:

```
cl /Mq cprog.c
```

After you have written your Borland C/C++ application program, compile your program and use the `-W` option of Borland C/C++ to create a Windows application. The following command compiles and links a Borland C/C++ Windows application named `cprog`:

```
bcc -W cprog.c
```

To run your application, choose the **Run** option from the **File** menu in the **Program Manager** window. Enter the path and name of the compiled program in the dialog box that appears.

Direct Entry with Visual Basic

See the *Visual Basic* section, earlier in this chapter, for information on calling the NI-488 functions with Visual Basic.

Include the file `niglobal.bas` in your application project file. You must create a file (for example, `dllfunc.bas`) that contains the direct entry prototypes of the functions used in your application program. Include the direct entry file in your application project file. Refer to the *NI-488.2 Function Reference Manual for DOS/Windows* for the Visual Basic direct entry syntax for NI-488 and NI-488.2 functions.

If you are using Microsoft Visual Basic 1.0, you must edit `niglobal.bas` to define the constants `true` and `false` as follows:

```
global const true  = -1
global const false =  0
```

In Microsoft Visual Basic 2.0 and higher, `true` and `false` are reserved words; therefore, these constants are no longer defined in `niglobal.bas`.

After you have written your Visual Basic application, choose the **Start** option from the **Run** menu to execute your program.

Chapter 4
Debugging Your Application

This chapter describes several ways to debug your application program.

Running wibtest

Before you run your application program, you should run the software diagnostic test, `wibtest`, that came with your NI-488.2 software. The `wibtest` program is an NI-488.2 application that makes calls to the driver. If `wibtest` passes, your GPIB hardware and NI-488.2 software are interacting correctly. The following paragraphs describe the messages you might receive while running `wibtest`, and how to resolve each problem. The term *gpibx* refers to one of the boards `gpib0`, `gpib1`, `gpib2`, and `gpib3`.

Presence Test of Driver

The `wibtest` program tests for the presence of the NI-488.2 driver `gpib.dll` and the NI-488.2 configuration information file `gpib.ini`. If `gpib.dll` is not in the Windows directory, a warning box pops up with the following text:

`File Error: Cannot find GPIB.DLL`

Press the **Close** button. `wibtest` displays the following error message when it fails:

`<<< No driver present for GPIBx. >>>`

To correct the problem, make sure that the file `gpib.dll` is in the Windows directory (usually `c:\windows`). The same error message is displayed if `gpib.ini` is not located in the Windows directory. If you are unable to locate `gpib.dll` or `gpib.ini`, you should reinstall the NI-488.2 software.

Presence Test of Board

The following error message appears if the board `gpib`*x* is not installed or if the software is not configured properly:

`<<< No board present for GPIBx. >>>`

If this message appears, you could have one of the following situations:

- The `Use this GPIB Interface` field in `wibconf` might be set to `no` for board `gpib`*x*. If you want to use the board, you must set this field to `yes`.

- The board might not be properly installed and configured. Refer to the getting started manual for detailed instructions.
- The software and hardware settings do not match. You can run `wibconf` to check the current configuration of the software.

GPIB Cables Connected

The following error message appears if a GPIB cable is connected to the board when you run `wibtest`:

```
Call(25) 'ibcmd " "' failed, ibsta (0x134) not what was expected
(0x8130)

Call(25) 'ibcmd " "' failed, expected ibsta (0x100) to have the
ERR bit set.
```

Disconnect all GPIB cables before trying the test again.

Running GPIBInfo

The `GPIBInfo` utility program is a simple diagnostic tool you can use to obtain information about the NI-488.2 software you are using and any GPIB interface boards in your system. This information helps you determine the capabilities of your NI-488.2 software and is also helpful if you need to call National Instruments for technical support.

If you run `GPIBInfo` with no specific parameters, the program displays software information such as the name and version of your GPIB software, the type of GPIB interface board and functions that you can use with the software, and whether or not you can use the HS488 high-speed protocol. `GPIBInfo` also displays information about each GPIB interface board installed in your system, including the name of the board, its Controller chip, the hardware settings, the type of functions that the board can use, and whether or not the board can use the HS488 high-speed communication protocol. The typical `GPIBInfo` output is as follows:

```
GPIBInfo (Sep 29 1993)
Copyright (c) 1993 National Instruments Corp.  All rights
reserved.

Software Information:
  The NI-488.2 Software for Windows 3 is loaded.
  You are running Version 2.5 for the AT-GPIB/TNT board.
  It supports both the NI-488 functions and the NI-488.2 routines.
  It supports the HS488 high-speed protocol.
```

```
Hardware Information:
  GPIB0: an AT-GPIB/TNT board using the TNT4882C chip.
         It supports both the NI-488 functions and NI-488.2
         routines.
         It supports the HS488 high-speed protocol.
         It uses base I/O address 0x2C0.
         It uses interrupt level 11.
         It uses DMA channel 5.
```

You can also run `GPIBInfo` by specifying the base I/O address of a GPIB board in your system. If `GPIBInfo` finds a board at the given address, it displays information about that board. This feature of `GPIBInfo` is useful in determining whether you have a GPIB board installed at a particular address. If you have a board installed at base I/O address 2C0 (hex), entering `gpibinfo 0x2c0` produces the following output:

```
GPIBInfo (Sep 29 1993)
Copyright (c) 1993 National Instruments Corp.  All rights
reserved.

The board at base I/O address 0x2C0 appears to be an AT-GPIB/TNT.
  It uses the TNT4882C GPIB Controller chip.
  It supports both the NI-488 functions and the NI-488.2 routines.
  It supports the HS488 high-speed protocol.
  The NI-488.2 software is configured to access this board as
  GPIB0.
```

Debugging with the Global Status Variables

After each function call to your NI-488.2 driver, `ibsta`, `iberr`, `ibcnt`, and `ibcntl` are updated before the call returns to your application. You should check for an error after each GPIB call. Refer to Chapter 3, *Developing Your Application*, for more information about how to use these variables within your program to automatically check for errors.

After you determine which GPIB call is failing and note the corresponding values of the global variables, refer to Appendix A, *Status Word Conditions*, and Appendix B, *Error Codes and Solutions*. These appendixes will help you interpret the state of the driver.

Debugging with wibic

If your application does not automatically check for and display errors, you can locate an error by using `wibic`. Simply issue the same functions or routines, one at a time as they appear in your application program. Because `wibic` returns the status values and error codes after each call, you should be able to determine which GPIB call is failing. For more information about `wibic`, refer to Chapter 5, *wibic–Windows Interface Bus Interactive Control Utility*.

After you determine which GPIB call is failing and note the corresponding values of the global variables, refer to Appendix A, *Status Word Conditions*, and Appendix B, *Error Codes and Solutions*. These appendixes will help you interpret the state of the driver.

Debugging with GPIB Spy

The NI-488.2 software includes GPIB Spy, an applications monitor utility that is useful as a debugging tool. You can use GPIB Spy to monitor the NI-488 and NI-488.2 calls made by your Windows application. You can configure GPIB Spy to suspend the execution of your application and display an information screen any time the error bit is set in `ibsta`. For more information about GPIB Spy, refer to Chapter 6, *GPIB Spy*.

GPIB Error Codes

Table 4-1 lists the GPIB error codes. Remember that the error variable is meaningful only when the ERR bit in the status variable is set. For a detailed description of each error and possible solutions, refer to Appendix B, *Error Codes and Solutions*.

Table 4-1. GPIB Error Codes

Error Mnemonic	iberr Value	Meaning
EDVR	0	Windows error
ECIC	1	Function requires GPIB board to be CIC
ENOL	2	No Listeners on the GPIB
EADR	3	GPIB board not addressed correctly
EARG	4	Invalid argument to function call
ESAC	5	GPIB board not System Controller as required
EABO	6	I/O operation aborted (timeout)
ENEB	7	Nonexistent GPIB board
EDMA	8	DMA error
EOIP	10	Asynchronous I/O in progress
ECAP	11	No capability for operation
EFSO	12	File system error
EBUS	14	GPIB bus error
ESTB	15	Serial poll status byte queue overflow
ESRQ	16	SRQ stuck in ON position
ETAB	20	Table problem

Configuration Errors

Several applications require customized configuration of the GPIB driver. For example, you might want to terminate reads on a special end-of-string character, or you might require secondary addressing. In these cases, you can use either the `wibconf` utility to permanently reconfigure the driver, or you can programmatically modify the driver while your application is running by using the NI-488 `ibconfig` function.

Note: ***To change settings other than base I/O address, interrupt level, or DMA channel, National Instruments recommends using*** `ibconfig` ***instead of running the*** `wibconf` ***utility***.

If your application uses `ibconfig`, it will always work regardless of the previous configuration of the driver. Refer to the description of `ibconfig` in the *NI-488.2 Function Reference Manual for DOS/Windows* for more information.

To test the configuration of your hardware, you can use the `ibdiag` program as described in your getting started manual.

Timing Errors

If your application fails, but the same calls issued in `wibic` are successful, your program might be issuing the NI-488.2 calls too quickly for your device to process and respond to them. This problem can also result in corrupted or incomplete data.

A well behaved IEEE 488 device should hold off handshaking and set the appropriate transfer rate. If your device is not well behaved, you can test for and resolve the timing error by single-stepping through your program and inserting finite delays between each GPIB call. One way to do this is to have your device communicate its status whenever possible. Although this method is not possible with many devices, it is usually the best option. Your delays will be controlled by the device and your application can adjust itself and work independently on any platform. Other delay mechanisms will probably cause varying delay times on different platforms.

Communication Errors

Repeat Addressing

Some devices require GPIB addressing before any GPIB activity. Devices adhering to the IEEE 488.2 standard should remain in their current state until specific commands are sent across the GPIB to change their state. You might need to configure your NI-488.2 driver to perform repeat addressing if your device does not remain in its currently addressed state. Refer to Chapter 8, *wibconf–Windows Interface Bus Configuration Utility*, or to the description of `ibconfig` (option `IbcREADDR`) in the *NI-488.2 Function Reference Manual for DOS/Windows* for more information about reconfiguring your software.

Termination Method

You should be aware of the data termination method that your device uses. By default, your NI-488.2 software is configured to send EOI on writes and terminate reads on EOI or a specific byte count. If you send a command string to your device and it does not respond, it might be because it does not recognize the end of the command. You might need to send a termination message such as <CR> <LF> after a write command as follows:

```
ibwrt(dev,"COMMAND\x0A\x0D",9);
```

Common Questions

Can I have the DOS and Windows drivers installed at the same time?

Yes, there is nothing wrong with installing both. However, it is better not to access them at the same time.

What do I do if `ibdiag` or `wibtest` fail with an error?

Refer to the *Running wibtest* section of this chapter and to the section about `ibdiag` in your getting started manual for specific information about what might cause these tests to fail.

How do I communicate with my instrument over the GPIB?

Refer to the documentation that came from the instrument manufacturer. The command sequences you use are totally dependent on the specific instrument. The documentation for each instrument should include the GPIB commands you need to communicate with it. In most cases, NI-488 device-level calls are sufficient for communicating with instruments. Refer to Chapter 3, *Developing Your Application*, for more information.

Can I use the NI-488 and NI-488.2 calls together in the same application?

Yes, you can mix NI-488 functions and NI-488.2 routines.

What can I do to check for errors in my GPIB application?

Examine the value of `ibsta` after each NI-488 or NI-488.2 call. If a call fails, the ERR bit of `ibsta` is set and an error code is stored in `ibcntl`. For more information about global status variables, refer to Chapter 3, *Developing Your Application.*

How do I use `wibic`?

You can use `wibic` to practice communication with your instrument, troubleshoot problems, and develop your application program. For instructions, refer to Chapter 5, *wibic–Windows Interface Bus Interactive Control Utility*.

What should I type into the pop-up window that appears when I start `wibconf`?

This pop-up window is prompting you for the drive and path where the `gpib.ini` file is located. If Windows is installed in the default directory (`c:\windows`), just press <Return>. If Windows is *not* installed in the default directory, enter the path and directory where Windows is installed so that `wibconf` can find the correct `gpib.ini` file to modify, for example `c:\win31\gpib.ini`.

How can I determine which type of GPIB board I have installed?

Run the `GPIBInfo` utility. If you run `GPIBInfo` without specifying any parameters, it returns information about the GPIB boards currently configured for use in your system. If you know the base I/O address of a GPIB interface board, you can enter it as a parameter for specific information. For example, `gpibinfo 2C0` returns information about the GPIB board at base I/O address 2C0.

How can I determine which version of the NI-488.2 software I have installed?

Run the `GPIBInfo` utility. If you run `GPIBInfo` without specifying any parameters, it provides information about the version of the NI-488.2 software currently installed.

Why can't I launch Windows after installing Windows 3.1 on top of my Windows 3.0 installation?

If the `system.ini` file contains conflicting information about which virtual devices to load, Windows fails to start. In this case, the `system.ini` file is probably trying to load two virtual DMA devices. The `system.ini` file should include the lines to load only one virtual DMA device. You can either delete unwanted lines or change them to comment lines by adding a semicolon at the beginning of the line.

The virtual DMA device is loaded in the `[386Enh]` section of the `system.ini` file. The correct one to use with Windows 3.1 is loaded by the following line:

```
device=*vdmad
```

The correct one to use with Windows 3.0 is loaded by the following line:

```
device=drive:\path\nivdmad.386
```

where `drive` and `path` describe the location of the `nivdmad.386` file on your hard disk (for example `c:\at-gpibw`).

What information should I have before I call National Instruments?

When you call National Instruments, you should have the results of the diagnostic tests `ibdiag` and `wibtest` along with the output from the `GPIBInfo` utility. Also, make sure you have filled out the technical support form in Appendix C, *Customer Communication*.

Chapter 5
wibic–Windows Interface Bus Interactive Control Utility

This chapter introduces you to `wibic`, the interactive control program that you can use to communicate with GPIB devices interactively.

Overview

With the `wibic` program, you communicate with the GPIB devices through functions you enter at the keyboard. For specific information about how to communicate with your particular device, refer to the manual that came with the device. You can use `wibic` to practice communicating with the instrument, troubleshoot problems, and develop your application program.

One way `wibic` helps you to learn about your instrument and to troubleshoot problems is by displaying the following information on your screen whenever you enter a command:

- The results of the status word (`ibsta`) in hexadecimal notation
- The mnemonic constant of each bit set in `ibsta`
- The mnemonic value of the error variable (`iberr`) if an error exists (the ERR bit is set in `ibsta`)
- The count value for each read, write, or command function
- The data received from your instrument

Example Using NI-488 Functions

This section shows how you might use `wibic` to test a sequence of NI-488 device function calls. You do not need to remember the parameters that each function takes. If you enter the function name only, `wibic` prompts you for the necessary parameters.

1. To run `wibic`, double-click on the **wibic** icon in the **GPIB** group in the **Program Manager**. A window appears with the following information displayed:

```
National Instruments
IEEE 488 Interface Bus Interactive Control Program (WIBIC)
Copyright (c) 1993 National Instruments Corp. Version 3.0 (Win16)
Version Date: May 28 1993 Version Time: 09:42:25
All rights reserved

Type 'help' for help or 'q' to quit

:
```

2. Use `ibdev` to find the device name which is assigned to your device in the `wibconf` program. The following example shows how you could use `ibdev` to open a device, assign it to access board `gpib0`, choose a primary address of 6 with no secondary address, set a timeout of 10 s, enable the END message, and disable the EOS mode:

```
:ibdev
    enter board index: 0
    enter primary address: 6
    enter secondary address: 0
    enter timeout: 13
    enter 'EOI on last byte' flag: 1
    enter end-of-string mode/byte: 0
id = 32256

ud0:
```

   You could also input all the same information with the `ibdev` command as follows:

```
:ibdev 0 6 0 13 1 0
id = 32256

ud0:
```

3. Clear the device as follows:

```
ud0: ibclr
[0100] (cmpl)
```

4. Write the function, range, and trigger source instructions to your device. Refer to the instrument's user manual for the command bytes that work with your instrument.

```
ud0: ibwrt
   enter string: "*RST; VAC; AUTO; TRIGGER 2; *SRE 16"
[0100] (cmpl)
count: 35
```

or

```
ud0: ibwrt "*RST; VAC; AUTO; TRIGGER 2; *SRE 16"
[0100] (cmpl)
count: 35
```

5. Trigger the device as follows:

```
ud0: ibtrg
[0100] (cmpl)
```

6. Wait for a timeout or for your device to request service. If the current timeout limit is too short, use `ibtmo` to change it. Use the `ibwait` command as follows:

```
ud0: ibwait
   enter wait mask: TIMO RQS
[0900] (rqs cmpl)
```

or

```
ud0: ibwait TIMO RQS
[0900] (rqs cmpl)
```

7. Read the serial poll status byte. This serial poll status byte varies depending on the device used.

```
ud0: ibrsp
[0100] (cmpl)
Poll: 0x40 (decimal : 64)
```

8. Use the read command to display the data on the screen both in hex values and their ASCII equivalents.

```
ud0: ibrd
   enter byte count: 18
[0100] (cmpl)
count: 18
4e 44 43 56 20 30 30 30     N D C V   0 0 0
2e 30 30 34 37 45 2b 30     . 0 0 4 7 E + 0
0a 0a                       . .
```

or

```
ud0: ibrd 18
[0100] (cmpl)
count: 18
4e 44 43 56 20 30 30 30     N D C V   0 0 0
2e 30 30 34 37 45 2b 30     . 0 0 4 7 E + 0
0a 0a                       . .
```

9. Place the device offline as follows:

```
ud0: ibonl
   enter value: 0
[0100] (cmpl)
```

or

```
ud0: ibonl 0
[0100] (cmpl)
```

10. Terminate the `wibic` program by entering `q` at the prompt.

wibic Syntax

When you enter commands in `wibic`, you can either include the parameters, or the program prompts you for values. Some commands require numbers as input values. Others might require you to input a string.

Number Syntax

You can enter numbers as hexadecimal, octal, or decimal integer.

Hexadecimal numbers–You must precede hex numbers by zero and x (for example, 0xD).

Octal numbers–You must precede octal numbers by zero only (for example, 015).

Decimal numbers–Enter the number only.

String Syntax

You can enter strings in the following formats.

ASCII character sequence–You must enclose the entire sequence in quotation marks.

Octal bytes–You must use a backslash character followed by the octal value. For example, octal 40 is represented by `\40`.

Hex bytes–You must use a backslash character and an `x` followed by the hex value. For example, hex 40 is represented by `\x40`.

Special Symbols–Some instruments require special termination or end-of-string (EOS) characters that indicate to the device that a transmission has ended. The two most common EOS characters are `\r` and `\n`. `\r` represents a carriage return character and `\n` represents a linefeed character. You can use these special characters to insert the carriage return and linefeed characters into a string, as in `"F3R5T1\r\n"`.

Address Syntax

Many of the NI-488.2 routines have an address or address list parameter. An address is a 16-bit representation of the GPIB address of a device. The primary address is stored in the low byte and the secondary address, if any, is stored in the high byte. For example, a device at primary address 6 and secondary address 0x67 has an address of 0x6706. A `NULL` address is represented as 0xffff.

wibic Syntax for NI-488 Functions

Table 5-1 and Table 5-2 summarize the syntax of NI-488 functions in `wibic`. `v` represents a number that you input. `string` represents a string that you input. For more information about the function parameters, use the `wibic` help feature or refer to the *NI-488.2 Function Reference Manual for DOS/Windows*.

Table 5-1. Syntax for Device-Level NI-488 Functions in wibic

Syntax	Description
`ibask mn`	Return configuration information where `mn` is a mnemonic for a configuration parameter or equivalent integer value
`ibbna brdname`	Change access board of device where `brdname` is symbolic name of new board
`ibclr`	Clear specified device
`ibconfig mn v`	Alter configurable parameters where `mn` is mnemonic for a configuration parameter or equivalent integer value
`ibdev v v v v v v`	Open an unused device `ibdev` parameters are `board id`, `pad`, `sad`, `tmo`, `eos`, `eot`
`ibeos v`	Change/disable EOS message
`ibeot v`	Enable/disable END message
`ibln v v`	Check for presence of device on the GPIB at `pad`, `sad`
`ibloc`	Go to local
`ibonl v`	Place device online or offline
`ibpad v`	Change primary address
`ibpct`	Pass control
`ibppc v`	Parallel poll configure
`ibrd v`	Read data where `v` is the bytes to read
`ibrda v`	Read data asynchronously where `v` is the bytes to read
`ibrdf flname`	Read data to file where `flname` is pathname of file to read
`ibrpp`	Conduct a parallel poll
`ibrsp`	Return serial poll byte
`ibsad v`	Change secondary address
`ibstop`	Abort asynchronous operation
`ibtmo v`	Change/disable time limit
`ibtrg`	Trigger selected device
`ibwait mask`	Wait for selected event where `mask` is a hex, octal, or decimal integer or a mask bit mnemonic
`ibwrt string`	Write data
`ibwrta string`	Write data asynchronously
`ibwrtf flname`	Write data from a file where `flname` is pathname of file to write

Table 5-2. Syntax for Board-Level NI-488 Functions in wibic

Syntax	Description
ibask mn	Return configuration information where mn is a mnemonic for a configuration parameter or equivalent integer value
ibcac v	Become Active Controller
ibcmd string	Send commands
ibcmda string	Send commands asynchronously
ibconfig mn v	Alter configurable parameters where mn is mnemonic for a configuration parameter or equivalent integer value
ibdma v	Enable/disable DMA
ibeos v	Change/disable EOS message
ibeot v	Enable/disable END message
ibevent	Return the oldest recorded event
ibfind udname	Return unit descriptor where udname is the symbolic name of a board (for example, or gpib0)
ibgts v	Go from Active Controller to standby
ibist v	Set/clear ist
iblines	Read the state of all GPIB control lines
ibln v v	Check for presence of device on the GPIB at pad, sad
ibloc	Go to local
ibonl v	Place device online or offline
ibpad v	Change primary address
ibppc v	Parallel poll configure
ibrd v	Read data where v is the bytes to read
ibrda v	Read data asynchronously where v is the bytes to read
ibrdf flname	Read data to file where flname is pathname of file to read
ibrpp	Conduct a parallel poll
ibrsc v	Request/release system control
ibrsv v	Request service
ibsad v	Change secondary address
ibsic	Send interface clear
ibsre v	Set/clear remote enable line
ibstop	Abort asynchronous operation
ibtmo v	Change/disable time limit
ibwait mask	Wait for selected event where mask is a hex, octal, or decimal integer or a mask bit mnemonic
ibwrt string	Write data
ibwrta string	Write data asynchronously
ibwrtf flname	Write data from a file where flname is pathname of file to write

wibic Syntax for NI-488.2 Routines

Table 5-3 summarizes the syntax of NI-488.2 routines in `wibic`. `v` represents a number and `string` represents a string. `address` represents an address, and `addrlist` represents a list of addresses separated by commas. For more information about the routine parameters, use the `wibic` help feature or refer to the *NI-488.2 Function Reference Manual for DOS/Windows*.

Table 5-3. Syntax for NI-488.2 Routines in wibic

Routine Syntax	Description
`AllSpoll addrlist`	Serial poll multiple devices
`DevClear address`	Clear a device
`DevClearList addrlist`	Clear multiple devices
`EnableLocal addrlist`	Enable local control
`EnableRemote addrlist`	Enable remote control
`FindLstn padlist v`	Find all Listeners
`FindRQS addrlist`	Find device asserting SRQ
`PassControl address`	Pass control to a device
`PPoll`	Parallel poll devices
`PPollConfig address v v`	Configure device for parallel poll
`PPollUnconfig addrlist`	Unconfigure device for parallel poll
`RcvRespMsg v v`	Receive response message
`ReadStatusByte address`	Serial poll a device
`Receive address v v`	Receive data from a device
`ReceiveSetup address`	Receive setup
`ResetSys addrlist`	Reset multiple devices
`Send address string v`	Send data to a device
`SendCmds string`	Send command bytes
`SendDataBytes buffer v`	Send data bytes
`SendIFC`	Send interface clear
`SendList addrlist string v`	Send data to multiple devices
`SendLLO`	Put devices in local lockout
`SendSetup addrlist`	Send setup
`Set 488.2 v`	Enter 488.2 mode for board `v`

(continues)

Table 5-3. Syntax for NI-488.2 Routines in wibic (Continued)

Routine Syntax	Description
SetRWLS addrlist	Put devices in remote with lockout state
TestSys addrlist	Causes multiple devices to perform self tests
TestSRQ	Test for service request
Trigger address	Trigger a device
TriggerList addrlist	Trigger multiple devices
WaitSRQ	Wait for service request

Status Word

In wibic, all NI-488 functions (except ibfind and ibdev) and NI-488.2 routines return the status word ibsta in two forms: a hex value in square brackets and a list of mnemonics in parentheses. In the following example, the status word is on the second line. It shows that the device function write operation completed successfully:

```
ud0: ibwrt "f2t3x"
[0100] (cmpl)
count: 5

ud0:
```

For more information about the status word, refer to Chapter 3, *Developing Your Application*.

Error Information

If an NI-488 function or NI-488.2 routine completes with an error, wibic displays the relevant error mnemonic. In the following example, the error condition EBUS has occurred during a data transfer.

```
ud0: ibwrt "f2t3x"
[8100] (err cmpl)
error: EBUS
count: 1

ud0:
```

In this example, the addressing command bytes could not be transmitted to the device. This indicates that either dev1 is powered off, or the GPIB cable is disconnected.

For a detailed list of the error codes and their meanings, refer to Chapter 4, *Debugging Your Application*.

Count

When an I/O function completes, `wibic` displays the actual number of bytes sent or received, regardless of the existence of an error condition.

If one of the addresses in an address list of an NI-488.2 routine is invalid, then the error is EARG and `wibic` displays the index of the invalid address as the count.

The count has a different meaning depending on which NI-488 function or NI-488.2 routine is called. Refer to the function descriptions in the *NI-488.2 Function Reference Manual for DOS/Windows* for the correct interpretation of the count return.

Common NI-488 Functions

ibfind

Use the `ibfind` function to open a board. The following example opens `gpib0`.

```
:ibfind gpib0
id = 32000

gpib0:
```

`id` is the unit descriptor of the board. The prompt `gpib0` indicates that the board is open.

Any name you use with the `ibfind` function must be a valid symbolic name in the driver. `gpib0` is the default name found in the driver. For more information about valid names, refer to Chapter 8, *wibconf–Windows Interface Bus Configuration Utility*.

ibdev

The `ibdev` command initializes a device descriptor with the input information.

With `ibdev`, you specify the following values:

- Access board for the device
- Primary address
- Secondary address

- Timeout setting
- EOT mode
- EOS mode

The following example shows `ibdev` opening an available device and assigning it to access `gpib0` (`board` = 0) with a primary address of 6 (`pad` = 6), a secondary address of hex 67 (`sad` = 0x67), a timeout of 10 s (`tmo`=13), the END message enabled (`eot` =1), and the EOS mode disabled (`eos` = 0).

```
:ibdev 0 6 0x67 13 1 0
id = 32256

ud0:
```

If you use `ibdev` without specifying parameters, `wibic` prompts you for the input parameters as shown in the following example:

```
:ibdev
       enter board index: 0
       enter primary address: 6
       enter secondary address: 0x67
       enter timeout: 13
       enter 'EOI on last byte' flag: 1
       enter end-of-string mode/byte: 0
id = 32256

ud0:
```

Three distinct errors can occur with the `ibdev` call:

- EDVR–No device is available, the board index entered refers to a nonexistent board (that is, not 0, 1, 2, or 3) or Windows cannot locate the `gpib.dll` file and/or `gpib.ini` file. The following example illustrates an EDVR error.

  ```
  :ibdev 4 6 0x67 7 1 0
  id = -1
  [8000] (err)
  error: EDVR (2)

  :
  ```

- ENEB–The board index entered refers to a known board (such as 0), but the driver cannot find the board. In this case, run `wibconf` to verify that the base address of the board is set correctly and that the `Use This GPIB Interface` field is set to `yes`.

- EARG–One of the last five parameters is an invalid value. The `ibdev` call returns with a new prompt and the EARG error (invalid function argument). If the `ibdev` call returns with an EARG error, you must identify which parameter is incorrect and use the appropriate command to correct it. In the following example, `pad` has an invalid value. You can correct it with an `ibpad` call as shown:

```
:ibdev 0 66 0x67 7 1 0
id = 32256
[8100] (err cmpl)
error: EARG

ud0: ibpad 6
previous value: 16
```

ibwrt

The `ibwrt` command sends data from one GPIB device to another. For example, to send the six character data string `F3R5T1` from the computer to a device, you enter the following string at the prompt as shown in the following example:

```
ud0: ibwrt "F3R5T1"
[0100] (cmpl)
count: 6
```

The returned status word contains the `cmpl` bit, which indicates a successful I/O completion. The byte count `6` indicates that all six characters were sent from the computer and received by the device.

ibrd

The `ibrd` command causes a GPIB device to receive data from another GPIB device. The following example acquires data from the device and displays it on the screen in hex format and in its ASCII equivalent, along with the status word and byte count.

```
ud0: ibrd 20
[2100] (end cmpl)
count: 18
4e 44 43 56 28 30 30 30    N D C V 9 0 0 0
2e 30 30 34 37 45 2b 30    . 0 0 4 7 E + 0
0d 0a                      . .
```

Common NI-488.2 Routines in wibic

Set

You must use the `set` command before you can use NI-488.2 routines in `wibic`. The syntax for this form of the `set` command is as follows:

```
set 488.2  n
```

where `n` represents a board number (for example, `n`=0 for `gpib0`).

The `488.2` prompt indicates that you are in NI-488.2 mode on board `n`. The following example shows how to enter into 488.2 mode on board `gpib0`.

```
set 488.2 0

488.2 (0):
```

Send and SendList

The `Send` routine sends data to a single GPIB device. You can use the `SendList` command to send data to multiple GPIB devices. For example, suppose you want to send the five character string `*IDN?` followed by the new line character with EOI. You want to send the message from the computer to the devices at primary address 2 and 17. To do this, enter the `SendList` command at the `488.2 (0)` prompt as shown in the following example:

```
488.2 (0): SendList 2, 17 "*IDN?" NLend
[0128] (cmpl cic tacs)
count: 6
```

The returned status word contains the `cmpl` bit, which indicates a successful I/O completion. The byte count `6` indicates that six characters, including the added new line, were sent from the computer and received by both devices.

Receive

The `Receive` routine causes the GPIB board to receive data from another GPIB device. The following example acquires 10 data bytes from the device at primary address 5. It stops receiving data when 10 characters have been received or when the END message is received. The acquired data is then displayed in hex format along with its ASCII equivalent. The `wibic` program also displays the status word and the count of transferred bytes.

```
488.2 (0): Receive 5 10 STOPend
[2124] (end cmpl cic lacs)
count: 5
48 65 6c 6c 6f      Hello
```

Auxiliary Functions

Table 5-4 summarizes the auxiliary functions that you can use in `wibic`.

Table 5-4. Auxiliary Functions in wibic

Function	Description
`set udname`	Select active device or board where `udname` is the symbolic name of the new device or board (for example, `dev1` or `gpib0`). Call `ibfind` or `ibdev` initially to open each device or board.
`help [option]`	Display help information where `option` is any NI-488 or NI-488.2 call. If you do not enter an `option`, a menu of options appears.
`!`	Repeat previous function.
`-`	Turn display off.
`+`	Turn display on.
`n* function`	Execute function `n` times where `function` represents the correct `wibic` function syntax.
`n* !`	Execute previous function `n` times.
`$ filename`	Execute indirect file where `filename` is the pathname of a file that contains `wibic` functions to be executed.
`print string`	Display string on screen where `string` is an ASCII character sequence, octal bytes, hex bytes, or special symbols.
`buffer option`	Set the type of display used for buffers. Valid options are `full`, `brief`, `ascii`, and `off`. The default is `full`.
`e`	Exit or quit.
`q`	Exit or quit.

Set (Select Device or Board)

You can use the `set` command to select 488.2 mode or to communicate with a different device. The following example switches communication from using NI-488.2 routines for `gpib0` to using a unit descriptor (`ud0`) previously acquired by an `ibdev` call.

```
488.2 (0): set ud0

ud0:
```

Help (Display Help Information)

The help feature displays a menu of topics to choose from. Each topic lists relevant functions and other information. You can access help for a specific NI-488 function or NI-488.2 routine by typing `help` followed by the call name (for example, `help ibwrt`). Help describes the function syntax for all NI-488 functions and NI-488.2 routines.

! (Repeat Previous Function)

The `!` function repeats the most recent function executed. The following example issues an `ibsic` command and then repeats that same command.

```
gpib0: ibsic
[0130] (cmpl cic atn)

gpib0: !
[0130] (cmpl cic atn)
```

- (Turn Display Off) and + (Turn Display On)

The `-` function turns off all screen output except for the prompt. This function is useful when you want to repeat any I/O function quickly without waiting for screen output to be displayed.

The `+` function turns the screen output on.

In the following example 24 consecutive letters of the alphabet are read from a device using three `ibrd` calls.

```
ud0: ibrd 8
[2100] (end cmpl)
count: 8
61 62 63 64 65 66 67 68    a b c d e f g h

ud0: -
```

```
ud0: ibrd 8

ud0: +

ud0: ibrd 8
[2100] (end cmpl)
count: 8
71 72 73 74 75 76 77 78    q r s t u v w x
```

n* (Repeat Function n Times)

The `n*` function repeats the execution of the specified function `n` times, where `n` is an integer. In the following example, the message `Hello` is sent five times to the device described by `ud0`.

```
ud0 : 5*ibwrt "Hello"
```

In the following example, the word `Hello` is sent five times, 20 times, and then 10 more times.

```
ud0: 5*ibwrt "Hello"
ud0: 20* !
ud0: 10* !
```

Notice that the multiplier (`*`) does not become part of the function name; that is, `ibwrt "Hello"` is repeated 20 times, not `5* ibwrt "Hello"`.

$ (Execute Indirect File)

The `$` function reads a specified file and executes the `wibic` functions listed in that file, in sequence, as if they were entered in that order from the keyboard. The following example executes the `wibic` functions listed in the file `userfile`.

```
gpib0: $ userfile
```

The following example repeats the operation three times.

```
gpib0: 3*$ userfile
```

The display mode that is in effect before this function was executed can be changed by functions in the indirect file.

Print (Display the ASCII String)

You can use the `print` function to echo a string to the screen. The following example shows how you can use ASCII or hex with the print command.

```
dev1: print "hello"
hello

dev1: print "and\r\n\x67\x6f\x6f\x64\x62\x79\x65"
and
goodbye
```

You can also use `print` to display comments from indirect files. The print string appears even if the display is suppressed with the `-` function.

Buffer (Set Buffer Display Mode)

You can set the type of display used for buffers to control how much of the buffer is displayed. `full` displays the entire buffer, `brief` displays the first and last eight bytes of the buffer, `ascii` displays the entire buffer in ASCII mode, and `off` displays none of the buffer.

Chapter 6
GPIB Spy

This chapter explains how to use the GPIB Spy utility to monitor and debug your application program.

Overview

GPIB Spy is a GPIB application monitor utility for Windows applications. It monitors the NI-488 and NI-488.2 calls made to the NI-488.2 driver for Windows, and records and displays information about the calls. You can use the GPIB Spy application to locate and analyze errors in your application or in the configuration of the NI-488.2 software.

Starting GPIB Spy

Start GPIB Spy by double-clicking on the **GPIB Spy** icon in the **GPIB** group in the **Program Manager**. This activates GPIB Spy and brings up the main **GPIB Spy** window. All NI-488 and NI-488.2 calls made while GPIB Spy is active are recorded in the main **GPIB Spy** window. GPIB Spy can be turned on and off by choosing **SpyOn** from the **Spy** menu. When GPIB Spy is on, the caption on the main **GPIB Spy** window is "GPIB Spy (recording)." When GPIB Spy is off, the caption is "GPIB Spy."

Figure 6-1 shows the main **GPIB Spy** window with several NI-488 calls recorded in it.

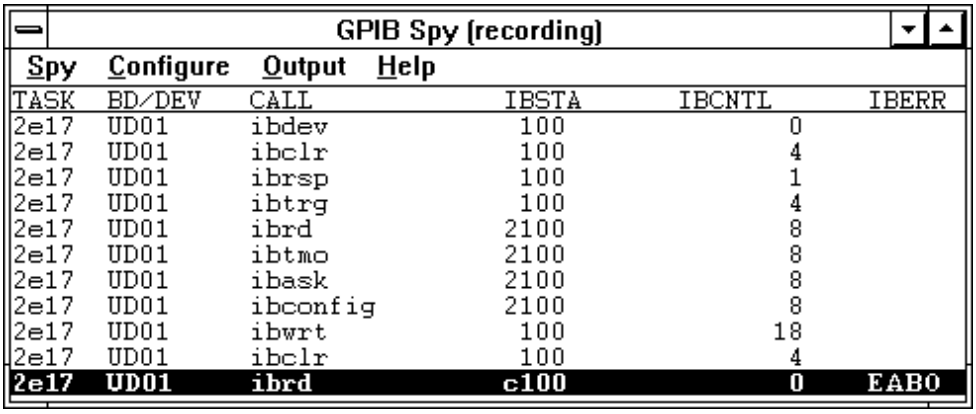

GPIB Spy (recording)
Spy Configure Output Help
TASK BD/DEV CALL IBSTA IBCNTL IBERR
2e17 UD01 ibdev 100 0
2e17 UD01 ibclr 100 4
2e17 UD01 ibrsp 100 1
2e17 UD01 ibtrg 100 4
2e17 UD01 ibrd 2100 8
2e17 UD01 ibtmo 2100 8
2e17 UD01 ibask 2100 8
2e17 UD01 ibconfig 2100 8
2e17 UD01 ibwrt 100 18
2e17 UD01 ibclr 100 4
2e17 UD01 ibrd c100 0 EABO

Figure 6-1. Main GPIB Spy Window

Exiting GPIB Spy

Exit GPIB Spy by choosing **Exit** from the **Spy** menu.

Caution: ***When you exit GPIB Spy, all of the recorded information is lost. See the section in this chapter entitled*** *GPIB Spy Output Options* ***if you want to avoid losing the information***.

Viewing Call Details

To see detailed information about calls recorded in GPIB Spy, use the **Call Details** window. You can activate the **Call Details** window in two different ways. If you double-click on any recorded call in the main **GPIB Spy** window, the call details for that particular call are displayed in the **Call Details** window. You can also select the **View Details** option from the **Spy** menu, which makes the **Call Details** window the top-level window.

You can use the **Call Details** window in two ways. If you check the **Continuously** item in the **View** menu, the **Call Details** window is updated as each new GPIB is recorded and you cannot scroll through the details for each recorded call. If you uncheck the **Continuously** item in the **View** menu, you can use the <Page Up> or <Page Down> keys to scroll through the call details of all recorded calls. If a call to the driver produces a GPIB error condition, the corresponding **Call Details** window displays an **iberr Help** button. Clicking on this button launches an interactive help utility which helps you identify and correct the problem.

Figure 6-2 shows the GPIB Spy **Call Details** window for the NI-488 function, `ibwrt`.

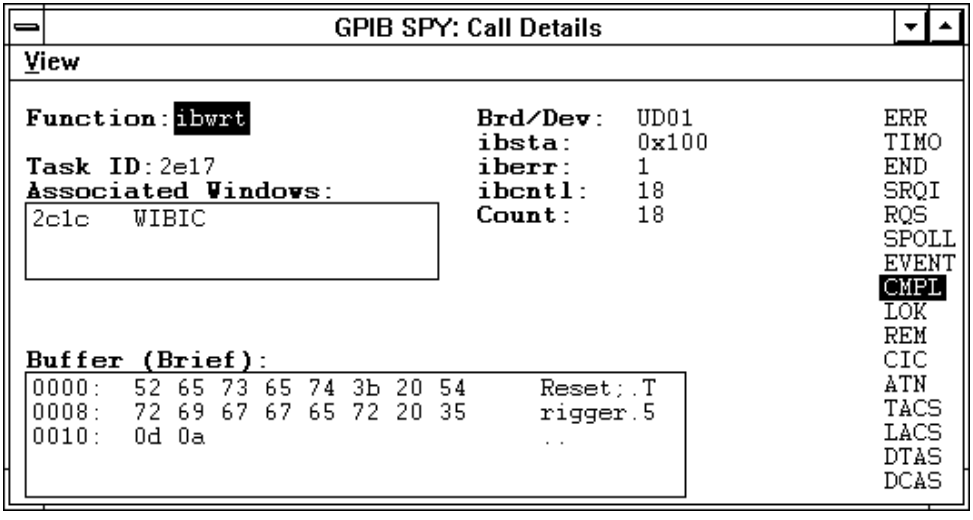


Figure 6-2. GPIB Spy Call Details Window for ibwrt

GPIB Spy Configuration Options

Calls To Highlight

You can choose for GPIB Spy to highlight certain calls based on bits set in `ibsta`. If you want to modify this option then select the **Calls to Highlight** menu item from the **Configure** menu. Use the dialog box to select which `ibsta` bits should cause a recorded call to be highlighted in the main **GPIB Spy** window. The default configuration of GPIB Spy is to highlight NI-488 and NI-488.2 calls which return with the ERR bit set in `ibsta`.

Calls To Trap On

You can configure GPIB Spy to suspend the execution of an application when one of the selected trap bits is set in `ibsta` for the NI-488 or NI-488.2 call that is completing. If you want to modify this option, select the **Calls to Trap On** menu item from the **Configure** menu. When a call causes a trap to occur, a dialog box pops up and gives a brief description of the trapped call. A dialog box causes all Windows applications currently running to be suspended until you click on the **OK** button. By default, GPIB Spy does not trap on any calls.

Buffer History

When GPIB Spy records a call such as `ibrd`, `ibwrt`, or `Send`, that includes a buffer of data, the **Call Details** window shows you the contents of the buffer. The default configuration of GPIB Spy is to store each I/O buffer in a brief format. If you want to modify this option then select the **Buffer History** menu item from the **Configure** menu. The buffer history has three settings: **brief**, **full**, and **none**. These settings determine how much of the buffer to save for the NI-488 or NI-488.2 call being recorded. Once a call is recorded, you cannot modify its buffer history. The **brief** buffer setting saves up to 64 bytes of buffer information. If the buffer is longer than 64 bytes, only the first 32 bytes and last 32 bytes of the buffer are saved. The **none** buffer setting saves no buffer information. The **full** buffer setting saves the entire I/O buffer. Note that the **full** buffer setting is likely to degrade the performance of your application and the whole windows system, especially if the application is transferring large GPIB buffers.

Set Highlight Color

You can use this option to configure the highlight color which is used to highlight information in the GPIB Spy windows. To change the highlight color, select the **Set Highlight Color** menu item from the **Configure** menu. This causes the standard Windows color selection dialog box to pop up. Choose a new color and click on **OK**. The default highlight color is red.

Store Configuration

You can use this option to make the current GPIB Spy configuration the default GPIB Spy configuration. If you want to set the default GPIB Spy configuration, select the **Store Configuration** menu item from the **Configure** menu.

GPIB Spy Output Options

To Screen

You can use this option to select the number of NI-488 and NI-488.2 calls that are displayed in GPIB Spy. By default GPIB Spy displays 100 calls. Only the most recent information is stored so that when the number of recorded calls exceeds the limit the oldest information is discarded. To change the number of calls displayed, select the **To Screen** menu item from the **Output** menu. Then enter the new number and click **OK** to save the change.

To File

You can use this option to specify an output file in which to save the information currently recorded in GPIB Spy. To save the information to a file, select the **To File** menu item from the **Output** menu. This causes the standard Windows file selection dialog box to pop up. Select a drive, directory and file name in which to save the information. When you click **OK**, the information recorded in GPIB Spy is saved to the specified file.

Check the **Update Continuously** check box in the dialog box if you want the GPIB Spy information to be saved as soon as it is recorded. The output file is opened, written, and closed as each NI-488 or NI-488.2 call is recorded. This option is useful if you are trying to debug a Windows application that causes the system to crash before you get an opportunity to close GPIB Spy. Use this option only if you need to because it is likely to slow down all Windows applications. In addition, if you are low on disk space, choosing this option could cause you to run out of disk space. To turn off the continuous update feature, unselect the **Update Continuously** check box in the file selection dialog box.

Clear Screen

This option clears the display and the information stored in GPIB Spy. You cannot retrieve information that has been cleared.

GPIB Spy Help

You can access online help for GPIB Spy by selecting the **GPIB Spy Help** menu item from the **Help** menu. All of the information covered in this chapter is available through the online help. The online help contains a thorough description of each of the features of GPIB Spy, as well as information to help you develop your Windows GPIB application.

Performance Considerations

Keep in mind that GPIB Spy can slow down the performance of your GPIB application and the whole windows system operating environment. If you record full I/O buffers, update to an output file continuously, or configure GPIB Spy to record the information on a large number of GPIB calls, it might have a significant effect on performance. For this reason, use GPIB Spy only while you are debugging your application or in situations in which performance is not critical.

Chapter 7
GPIB Programming Techniques

This chapter describes techniques for using some NI-488 functions and NI-488.2 routines in your application program.

For more detailed information about each function or routine, refer to the *NI-488.2 Function Reference Manual for DOS/Windows*.

Termination of Data Transfers

GPIB data transfers are terminated either when the GPIB EOI line is asserted with the last byte of a transfer or when a preconfigured end-of-string (EOS) character is transmitted. By default, the NI-488.2 driver asserts EOI with the last byte of writes and the EOS modes are disabled.

You can use the `ibeot` function to enable or disable the end of transmission (EOT) mode. If EOT mode is enabled, the NI-488.2 driver asserts the GPIB EOI line when the last byte of a write is sent out on the GPIB. If it is disabled, the EOI line is *not* asserted with the last byte of a write.

You can use the `ibeos` function to enable, disable, or configure the EOS modes. EOS mode configuration includes the following information:

- A 7-bit or 8-bit EOS byte
- EOS comparison method–This indicates whether the EOS byte has seven or eight significant bits. For a 7-bit EOS byte, the eighth bit of the EOS byte is ignored.
- EOS write method–If this is enabled, the NI-488.2 driver automatically asserts the GPIB EOI line when the EOS byte is written to the GPIB. If the buffer passed into an `ibwrt` call contains five occurrences of the EOS byte, the EOI line is asserted as each of the five EOS bytes are written to the GPIB. If an `ibwrt` buffer does not contain an occurrence of the EOS byte, the EOI line is not asserted (unless the EOT mode is enabled, in which case the EOI line is asserted with the last byte of the write).
- EOS read method–If this is enabled, the NI-488.2 driver terminates `ibrd`, `ibrda`, and `ibrdf` calls when the EOS byte is detected on the GPIB, when the GPIB EOI line is asserted, or when the specified count is reached. If the EOS read method is disabled, `ibrd`, `ibrda`, and `ibrdf` calls terminate only when the GPIB EOI line is asserted or the specified count has been read.

You can use the `ibconfig` function to configure the software to inform you whether or not the GPIB EOI line was asserted when the EOS byte was read. Use the `IbcEndBitIsNormal` option to configure the software to report only the END bit in `ibsta` when the GPIB EOI line is asserted. By default, the NI-488.2 driver reports END in `ibsta` when either the EOS byte is read in or the EOI line is asserted during a read.

High-Speed Data Transfers (HS488)

National Instruments has designed a high-speed data transfer protocol for IEEE 488 called *HS488*. This protocol increases performance for GPIB reads and writes up to 8 Mbytes/s, depending on your system.

HS488 is a superset of the IEEE 488 standard; thus, you can mix IEEE 488.1, IEEE 488.2, and HS488 devices in the same system. If HS488 is enabled, the TNT4882C hardware implements high-speed transfers automatically when communicating with HS488 instruments. To determine whether your GPIB interface board has the TNT4882C hardware, use the `GPIBInfo` utility. If you attempt to enable HS488 on a GPIB board that does not have the TNT4882C chip, the error ECAP is returned.

Enabling HS488

To enable HS488 for your GPIB board, use the `ibconfig` function (option `IbcHSCableLength`). The value passed to `ibconfig` should specify the number of meters of cable in your GPIB configuration. If you specify a cable length that is much smaller than what you actually use, the transferred data could become corrupted. If you specify a cable length longer than what you actually use, the data is transferred successfully, but more slowly than if you specified the correct cable length.

In addition to using `ibconfig` to configure your GPIB board for HS488, the Controller-In-Charge must send out GPIB command bytes (interface messages) to configure other devices for HS488 transfers.

If you are using device-level calls, the NI-488.2 software automatically sends the HS488 configuration message to devices. If you enabled the HS488 protocol in `wibconf`, the NI-488.2 software sends out the HS488 configuration message when you use `ibdev` to bring a device online. If you call `ibconfig` to change the GPIB cable length, the NI-488.2 software sends out the HS488 message again the next time you call a device-level function.

If you are using board-level functions or NI-488.2 routines and you want to configure devices for high-speed, you must send the HS488 configuration messages using `ibcmd` or `SendCmds`. The HS488 configuration message is made up of two GPIB command bytes. The first byte, the Configure Enable (CFE) message (hex 1F), places all HS488 devices into their configuration mode. Non-HS488 devices should ignore this message. The second byte is a GPIB secondary command that indicates the number of meters of

cable in your system. It is called the Configure (CFGn) message. Because HS488 can operate only with cable lengths of 1 to 15 meters, only CFGn values of 1 through 15 (hex 61 through 6F) are valid. If the cable length was configured correctly in `wibconf`, you can determine how many meters of cable are in your system by calling `ibask` (option `IbaHSCableLength`) in your application program. For CFE and CFGn messages, refer to Appendix A, *Multiline Interface Messages*, in the *NI-488.2 Function Reference Manual for DOS/Windows*.

System Configuration Effects on HS488

Maximum data transfer rates can be limited by your host computer and GPIB system setup. For example, even though the theoretical maximum transfer rate with HS488 is 8 Mbytes/s, the maximum transfer rate obtainable on PC compatible computers with an ISA bus is 2 Mbytes/s. The same IEEE 488 cabling constraints for a 350 ns T1 delay apply to HS488. As you increase the amount of cable in your GPIB configuration, the maximum data transfer rate using HS488 decreases. For example, two HS488 devices connected by two meters of cable can transfer data faster than three HS488 devices connected by four meters of cable.

Waiting for GPIB Conditions

You can use the `ibwait` function to obtain the current `ibsta` value or to suspend your application until a specified condition occurs on the GPIB. If you use `ibwait` with a parameter of zero, it immediately updates `ibsta` and returns. If you want to use `ibwait` to wait for one or more events to occur, then pass a wait mask to the function. The wait mask should always include the TIMO event; otherwise, your application is suspended indefinitely until one of the wait mask events occurs.

Device-Level Calls and Bus Management

The NI-488 device-level calls are designed to perform all of the GPIB management for your application program. However, the NI-488.2 driver can handle bus management only when the GPIB interface board is CIC (Controller-In-Charge). Only the CIC is able to send command bytes to the devices on the bus to perform device addressing or other bus management activities. Use one of the following methods to make your GPIB board the CIC:

- If your GPIB board is configured as the System Controller (default), it automatically makes itself the CIC by asserting the IFC line the first time you make a device-level call.

- If your setup includes more than one Controller, or if your GPIB interface board is not configured as the System Controller, use the CIC Protocol method. To use the protocol, issue the `ibconfig` function (option `IbcCICPROT`) or use the `wibconf` configuration utility to activate the CIC protocol. If the interface board is not CIC and you make a device-level call with the CIC Protocol enabled, the following sequence occurs:

  1. The GPIB interface board asserts the SRQ line.
  2. The current CIC serial polls the board.
  3. The interface board returns a response byte of hex 42.
  4. The current CIC passes control to the GPIB board.

  If the current CIC does not pass control, the NI-488.2 driver returns the ECIC error code to your application. This error can occur if the current CIC does not understand the CIC Protocol. If this happens, you could send a device-specific command requesting control for the GPIB board. Then use a board-level `ibwait` command to wait for CIC.

Talker/Listener Applications

Although designed for Controller-In-Charge applications, you can also use the NI-488.2 software in most non-Controller situations. These situations are known as Talker/Listener applications because the interface board is not the GPIB Controller. A typical Talker/Listener application waits for commands from the Controller and responds as appropriate. The following paragraphs describe some programming techniques for Talker/Listener applications.

Waiting for Messages from the Controller

A Talker/Listener application typically uses `ibwait` with a mask of 0 to monitor the status of the interface board. Then, based on the status bits set in `ibsta`, the application takes whatever action is appropriate. For example, the application could monitor the status bits TACS (Talker Active State) and LACS (Listener Active State) to determine when to send data to or receive data from the Controller. The application could also monitor the DCAS (Device Clear Active State) and DTAS (Device Trigger Active State) bits to determine if the Controller has sent the device clear (DCL or SDC) or trigger (GET) messages to the interface board. If the application detects a device clear from the Controller, it might reset the internal state of message buffers. If it detects a trigger message from the Controller, the application might begin an operation such as taking a voltage reading if the application is actually acting as a voltmeter.

Using the Event Queue

Some applications need to know the order in which certain messages are sent by the Controller. To monitor the ordering of these messages, your application program must enable the EVENT bit, using `ibconfig` (option `IbcEventQueue`). When the EVENT bit is enabled, the DCAS and DTAS bits are no longer activated. Instead, all DCAS and DTAS messages are stored in a queue, in the order that they are received. The event queue also stores interface clear (IFC) messages. When the queue contains some information, the NI-488.2 software sets the EVENT bit in `ibsta`. When the application program detects EVENT, it can call the function `ibevent` to retrieve the first event that occurred. Retrieving events from the queue ensures that the application can respond to device clear, device trigger, and interface clear messages in the correct order.

Requesting Service

Another type of event that might be important in a Talker/Listener application is the serial poll. A Talker/Listener application can call `ibrsv` with a serial poll response byte when it needs to request service from the Controller. If the application needs to know when the Controller has read the serial poll response byte, it can enable the SPOLL bit in `ibsta` using `ibconfig`, option `IbcSPollBit`. The NI-488.2 software sets the SPOLL bit when the Controller serial polls the board.

Serial Polling

You can use serial polling to obtain specific information from GPIB devices when they request service. When the GPIB SRQ line is asserted, it signals the Controller that a service request is pending. The Controller must then determine which device asserted the SRQ line and respond accordingly. The most common method for SRQ detection and servicing is the serial poll. This section describes how you can set up your application to detect and respond to service requests from GPIB devices.

Service Requests from IEEE 488 Devices

IEEE 488 devices request service from the GPIB Controller by asserting the GPIB SRQ line. When the Controller acknowledges the SRQ, it serial polls each open device on the bus to determine which device requested service. Any device requesting service returns a status byte with bit 6 set and then unasserts the SRQ line. Devices not requesting service return a status byte with bit 6 cleared. Manufacturers of IEEE 488 devices use lower order bits to communicate the reason for the service request or to summarize the state of the device.

Service Requests from IEEE 488.2 Devices

The IEEE 488.2 standard refined the bit assignments in the status byte. In addition to setting bit 6 when requesting service, IEEE 488.2 devices also use two other bits to specify their status. Bit 4, the Message Available bit (MAV), is set when the device is ready to send previously queried data. Bit 5, the Event Status bit (ESB), is set if one or more of the enabled IEEE 488.2 events occurs. These events include power-on, user request, command error, execution error, device-dependent error, query error, request control, and operation complete. The device can assert SRQ when ESB or MAV are set, or when a manufacturer-defined condition occurs.

Automatic Serial Polling

You can enable automatic serial polling if you want your application to conduct a serial poll automatically any time the SRQ line is asserted. The autopolling procedure occurs as follows:

1. To enable autopolling, use the configuration utility, `wibconf`, or the configuration function, `ibconfig` with option `IbcAUTOPOLL`. (Autopolling is enabled by default.)

2. When the SRQ line is asserted, the driver automatically serial polls the open devices.

3. Each positive serial poll response (bit 6 or hex 40 is set) is stored in a queue associated with the device that sent it. The RQS bit of the device status word, `ibsta`, is set.

4. The polling continues until SRQ is unasserted or an error condition is detected.

5. To empty the queue, use the `ibrsp` function. `ibrsp` returns the first queued response. Other responses are read in first-in-first-out (FIFO) fashion. If the RQS bit of the status word is not set when `ibrsp` is called, a serial poll is conducted and returns whatever response is received. You should empty the queue as soon as an automatic serial poll occurs, because responses might be discarded if the queue is full.

6. If the RQS bit of the status word is still set after `ibrsp` is called, the response byte queue contains at least one more response byte. If this happens, you should continue to call `ibrsp` until RQS is cleared.

Stuck SRQ State

If autopolling is enabled and the GPIB interface board detects an SRQ, the driver serial polls all open devices connected to that board. The serial poll continues until either SRQ unasserts or all the devices have been polled.

If no device responds positively to the serial poll, or if SRQ remains in effect because of a faulty instrument or cable, a *stuck SRQ* state is in effect. If this happens during an `ibwait` for RQS, the driver reports the ESRQ error. If the *stuck SRQ* state happens, no further polls are attempted until an `ibwait` for RQS is made. When `ibwait` is issued, the *stuck SRQ* state is terminated and the driver attempts a new set of serial polls.

Autopolling and Interrupts

If autopolling and interrupts are both enabled, the NI-488.2 software can perform autopolling after any device-level NI-488 call as long as no GPIB I/O is currently in progress. This means that an automatic serial poll can occur even when your application is not making any calls to the NI-488.2 software. Autopolling can also occur when a device-level `ibwait` for RQS is in progress. Autopolling is not allowed whenever an application calls a board-level NI-488 function or any NI-488.2 routine, or the *stuck SRQ* (ESRQ) condition occurs.

If autopolling is enabled and interrupts are disabled, you can use autopolling in the following situations only:

- During a device-level `ibwait` for RQS
- Immediately after a device-level NI-488 function is completed, before control is returned to the application program

SRQ and Serial Polling with NI-488 Device Functions

You can use the device-level NI-488 function `ibrsp` to conduct a serial poll. `ibrsp` conducts a single serial poll and returns the serial poll response byte to the application program. If automatic serial polling is enabled, the application program can use `ibwait` to suspend program execution until RQS appears in the status word, `ibsta`. The program can then call `ibrsp` to obtain the serial poll response byte.

The following example illustrates the use of the `ibwait` and `ibrsp` functions in a typical SRQ servicing situation when automatic serial polling is enabled.

```
#include "decl.h"

char GetSerialPollResponse ( int DeviceHandle )
{
       char SerialPollResponse = 0;

       ibwait ( DeviceHandle, TIMO | RQS );
```

```
        if ( ibsta & RQS )  {
                printf ( "Device asserted SRQ.\n" );
                /*  Use ibrsp to retrieve the serial poll
                    response.  */
                ibrsp ( DeviceHandle, &SerialPollResponse );
        }
        return SerialPollResponse;
}
```

SRQ and Serial Polling with NI-488.2 Routines

The NI-488.2 software includes a set of NI-488.2 routines that you can use to conduct SRQ servicing and serial polling. Routines pertinent to SRQ servicing and serial polling are `AllSpoll`, `FindRQS`, `ReadStatusByte`, `TestSRQ`, and `WaitSRQ`.

`AllSpoll` can serial poll multiple devices with a single call. It places the status bytes from each polled instrument into a predefined array. Then you must check the RQS bit of each status byte to determine whether that device requested service.

`ReadStatusByte` is similar to `AllSpoll`, except that it serial polls only a single device. It is also analogous to the device-level NI-488 `ibrsp` function.

`FindRQS` serial polls a list of devices until it finds a device that is requesting service or until it has polled all of the specified devices. The routine returns the index and status byte value of the device requesting service.

`TestSRQ` determines whether the SRQ line is asserted or unasserted, and returns to the program immediately.

`WaitSRQ` is similar to `TestSRQ`, except that `WaitSRQ` suspends the application program until either SRQ is asserted or the timeout period is exceeded.

The following examples use NI-488.2 routines to detect SRQ and then determine which device requested service. In these examples three devices are present on the GPIB at addresses 3, 4, and 5, and the GPIB interface is designated as bus index 0. The first example uses `FindRQS` to determine which device is requesting service and the second example uses `AllSpoll` to serial poll all three devices. Both examples use `WaitSRQ` to wait for the GPIB SRQ line to be asserted.

Note: ***Automatic serial polling is not used in these examples because you cannot use it with NI-488.2 routines***.

Example 1: Using FindRQS

This example illustrates the use of `FindRQS` to determine which device is requesting service.

```
void GetASerialPollResponse ( char *DevicePad,
                              char *DeviceResponse )
{
    char SerialPollResponse = 0;
    int WaitResult;
    Addr4882_t Addrlist[4] = {3,4,5,NOADDR};

    WaitSRQ (0, &WaitResult);

    if (WaitResult) {
       printf ("SRQ is asserted.\n");

/* Use FindRQS to find a device that requested service. */

      FindRQS ( 0, AddrList, &SerialPollResponse );
      if (!(ibsta & ERR))  {
         printf ("Device at pad %x returned byte %x.\n",
                  AddrList[ibcnt],(int) SerialPollResponse);
         *DevicePad = AddrList[ibcnt];
         *DeviceResponse = SerialPollResponse;
      }
   }

   return;
}
```

Example 2: Using AllSpoll

This example illustrates the use of `AllSpoll` to serial poll three devices.

```
void GetAllSerialPollResponses ( Addr4882_t AddrList[], short
ResponseList[] )
{
   int WaitResult;

   WaitSRQ (0, &WaitResult);

   if ( WaitResult ) {
      printf ( "SRQ is asserted.\n" );
```

```
/* Use Allspoll to serial poll all the devices at once. */

     AllSpoll ( 0, AddrList, ResponseList );
     if (!(ibsta & ERR))  {
        for ( i = 0; AddrList[i] != NOADDR; i++ )  {
           printf ("Device at pad %x returned byte %x.\n",
                    AddrList[i], ResponseList[i] );
        }
     }
  }

  return;
}
```

Parallel Polling

Although parallel polling is not widely used, it is a useful method for obtaining the status of more than one device at the same time. The advantage of a parallel poll is that it can easily check up to eight individual devices at once. In comparison, eight separate serial polls would be required to check eight devices for their serial poll response bytes.

Implementing a Parallel Poll

You can implement parallel polling with either NI-488 functions or NI-488.2 routines. If you use NI-488.2 routines to execute parallel polls, you do not need extensive knowledge of the parallel polling messages. However, you should use the NI-488 functions for parallel polling when the GPIB board is not the Controller and must configure itself for a parallel poll and set its own individual status bit (`ist`).

Parallel Polling with NI-488 Functions

Follow these steps to implement parallel polling using NI-488 functions. Each step contains example code.

1. Configure the device for parallel polling using the `ibppc` function, unless the device can configure itself for parallel polling.

   `ibppc` requires an 8-bit value to designate the data line number, the `ist` sense, and whether or not the function configures or unconfigures the device for the parallel poll. The bit pattern is as follows:

   0 1 1 E S D2 D1 D0

   E is 1 to disable parallel polling and 0 to enable parallel polling for that particular device.

S is 1 if the device is to assert the assigned data line when `ist` = 1, and 0 if the device is to assert the assigned data line when `ist` = 0.

D2 through D0 determine the number of the assigned data line. The physical line number is the binary line number plus one. For example, DIO3 has a binary bit pattern of 010.

The following example code configures a device for parallel polling using NI-488 functions. The device asserts DIO7 if its `ist` = 0.

In this example, the `ibdev` command is used to open a device that has a primary address of 3, has no secondary address, has a timeout of 3 s, asserts EOI with the last byte of a write operation, and has EOS characters disabled.

```
#include "decl.h"
char ppr;

dev = ibdev(0,3,0,T3s,1,0);

/* Pass the binary bit pattern, 0110 0110 or hex 66, to
   ibppc */

ibppc(dev, 0x66);
```

If the GPIB interface board configures itself for a parallel poll, you should still use the `ibppc` function. Pass the board index or a board unit descriptor value as the first argument in `ibppc`. In addition, if the individual status bit (`ist`) of the board needs to be changed, use the `ibist` function.

In the following example, the GPIB board is to configure itself to participate in a parallel poll. It asserts DIO5 when `ist` = 1 if a parallel poll is conducted.

```
ibppc(0, 0x6C);
ibist(0, 1);
```

2. Conduct the parallel poll using `ibrpp` and check the response for a certain value. The following example code performs the parallel poll and compares the response to hex 10, which corresponds to DIO5. If that bit is set, the `ist` of the device is 1.

```
ibrpp(dev, &ppr);
if (ppr & 0x10) printf("ist = 1\n");
```

3. Unconfigure the device for parallel polling with `ibppc`. Notice that any value having the parallel poll disable bit set (bit 4) in the bit pattern disables the configuration, so you can use any value between hex 70 and 7E.

```
ibppc(dev, 0x70);
```

Parallel Polling with NI-488.2 Routines

Follow these steps to implement parallel polling using NI-488.2 routines. Each step contains example code.

1. Configure the device for parallel polling using the `PPollConfig` routine, unless the device can configure itself for parallel polling. The following example configures a device at address 3 to assert data line 5 (DIO5) when its `ist` value is 1.

```
#include "decl.h"
char response;
Addr4882_t AddressList[2];

/* The following command clears the GPIB.   */

SendIFC(0);

/* The value of sense is compared with the ist bit of the
   device and determines whether the data line is asserted. */

PPollConfig(0,3,5,1);
```

2. Conduct the parallel poll using `PPoll`, store the response, and check the response for a certain value. In the following example, because DIO5 is asserted by the device if `ist` = 1, the program checks bit 4 (hex 10) in the response to determine the value of `ist`.

```
PPoll(0, &response);

/* If response has bit 4 (hex 10) set, the ist bit of the
   device at that time is equal to 1.  If it does not appear,
   the ist bit is equal to 0. Check the bit in the following
   statement. */

if (response & 0x10) {
   printf("The ist equals 1.\n");
}
else {
   printf("The ist equals 0.\n");
}
```

3. Unconfigure the device for parallel polling using the `PPollUnconfig` routine as shown in the following example. In this example, the `NOADDR` constant must appear at the end of the array to signal the end of the address list. If `NOADDR` is the only value in the array, all devices receive the parallel poll disable message.

```
AddressList[0] = 3;
AddressList[1] = NOADDR;
PPollUnconfig(0, AddressList);
```

Chapter 8
wibconf–Windows Interface Bus Configuration Utility

This chapter contains a description of `wibconf`, the software configuration program you can use to configure the NI-488.2 software for Windows.

Overview

The `wibconf` utility is a screen-oriented, interactive program you can use to view or modify the configuration parameters of your GPIB interface boards and the GPIB devices connected to them. Usually, you use `wibconf` to modify the hardware settings of your GPIB interface boards.

The `wibconf` utility can read in and display configuration parameters for the NI-488.2 driver file on disk. You can then save the changes back to the disk file. `wibconf` also has a supplementary batch mode which can be used for configuring in a non-interactive fashion.

Instead of using `wibconf`, you can configure your driver programmatically by using the `ibconfig` function to alter any of the board or device characteristics while your program is running. If you use dynamic configuration, you do not need to run `wibconf` before you start your application. Also, you can run your application on any computer with the appropriate NI-488.2 software regardless of its configuration because the application configures the driver as necessary. Programmatic configuration is the preferred method of GPIB application development.

Starting wibconf

The simplest way to use `wibconf` is to double-click on the `wibconf` icon in your **GPIB** group in the Windows **Program Manager**. Windows pops up a dialog box labeled **Parameters**.

By default, `wibconf` looks for the NI-488.2 software configuration file, `gpib.ini`, in the `c:\windows` directory. If you installed Windows in a directory other than `c:\windows`, specify the full path to the `gpib.ini` file in the **Parameters** box. For example, if you installed Windows in the directory `d:\win3`, type `d:\win3\gpib.ini` in the **Parameters** dialog box.

If you installed Windows in the default directory, simply press <Return>.

Table 8-1 lists the options you can select when you start `wibconf`.

Table 8-1. Options for Starting wibconf

wibconf Option	Action
driver	Configure a specific driver. `wibconf` configures the given driver configuration file (for example, `wibconf d:\win3\gpib.ini`).
`-h`	Help. This option causes `wibconf` to display a summary of options.
`-b` *filename*	Batch mode configuration. `wibconf` is run in batch mode using the configuration information contained in the given file (for example, `wibconf -b gpib.cfg`). See the *wibconf Batch Mode* section later in this chapter.
`-e`	Expert mode. This option disables `wibconf` warning messages when you exit `wibconf`. See the *Exiting wibconf* section later in this chapter.
`-m`	Monochrome mode. This option causes `wibconf` to run in monochrome mode even if you have a color monitor.
`-vb`	Video access through BIOS. This option causes `wibconf` to use the system BIOS routines when writing to the screen. This is slower than the default of direct screen access, but is more compatible with certain nonstandard systems.

Upper and Lower Levels of wibconf

`wibconf` operates at both an upper and a lower level. The upper level consists of the device map that gives a graphical picture of the GPIB system. The lower level consists of screens describing the individual boards and devices that make up the system.

Upper-Level Device Map

Figure 8-1 shows the upper level of `wibconf`.

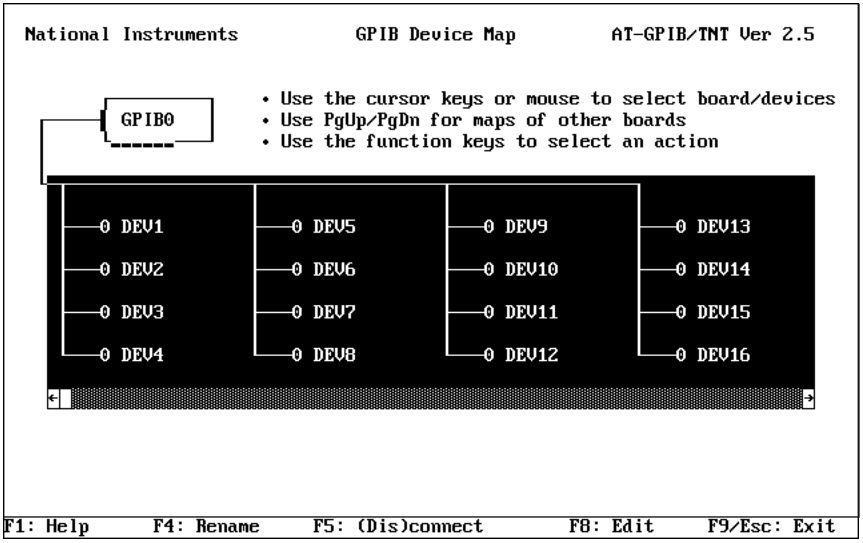


Figure 8-1. Upper Level of wibconf

As shown in Figure 8-1, the upper-level screen of `wibconf` displays the names of all devices controlled by the driver. It also indicates which devices, if any, are accessed through each interface board. You can move around the map by using the cursor control keys or the mouse.

The following options are available at the upper level.

- Device maps of the boards
- Help
- Rename
- (Dis)connect
- Edit
- Exit

Device Maps of the Boards

Use <PageUp> or <PageDown> to toggle between the device maps for the different GPIB interface boards. These boards are referred to as *access boards*. The maps show which devices are assigned to each board.

Help

Use the function key <F1> to access the comprehensive, online help feature of `wibconf`. The help information describes the functions and common terms associated with the upper level of `wibconf`.

Rename

Use the function key <F4> to rename a device. Move to the device you want to rename by using the cursor control keys. Press the <F4> key and enter the new name of the device. The device name may contain up to eight characters in lowercase or uppercase.

The following restrictions apply when renaming a device:

- Extensions (.xxx) are not allowed
- As specified by DOS, the device name cannot use the following characters:

```
.       "       /       \       [       ]       :
|       <       >       +       =       ;       ,
```

- Do not use the reserved names `con` or `nul` for your device
- Do not give GPIB device names the same names as files, directories, or subdirectories. If you name a GPIB device `pltr` and your file system contains the file `pltr.dat` or a subdirectory `pltr`, a conflict results.
- The names of the access boards `gpib0`, `gpib1`, `gpib2`, and `gpib3` are fixed and connot be changed.

(Dis)connect

Use the function key <F5> to connect or disconnect a device from a particular access board. Move the cursor to the device that you want to connect or disconnect by using the cursor control keys and then press the <F5> key.

Edit Use the function key <F8> or the <Enter> key to edit or examine the characteristics of a particular board or device. Move to the board or device that you want to edit using the cursor control keys and then press the <F8> key. This step puts you in the lower level of `wibconf` and lists the characteristics for the particular board or device that you want to edit. To exit edit, press the function key <F9> or <Escape> to return to the upper level.

Exit Use the function key <F9> or <Escape> to exit `wibconf`. If you have made changes, `wibconf` prompts you to save the changes before exiting. For more information, refer to the *Exiting wibconf* section later in this chapter.

Lower-Level Device/Board Characteristics

The lower-level screens of `wibconf` display the currently defined values for characteristics of a device or board such as addressing and timeout information, as shown in Figure 8-2.

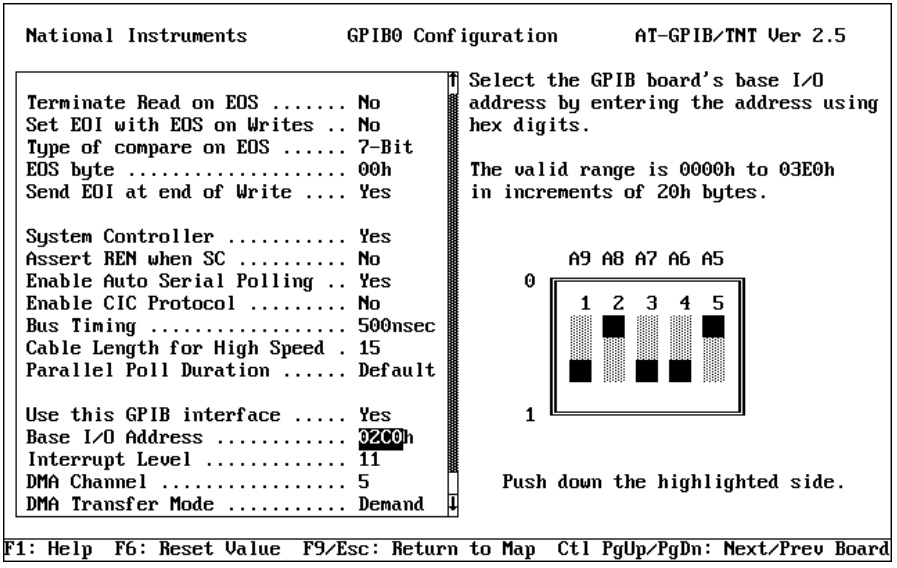

National Instruments GPIB0 Configuration AT-GPIB/TNT Ver 2.5
Terminate Read on EOS ..... No
Set EOI with EOS on Writes .. No
Type of compare on EOS 7-Bit
EOS byte 00h
Send EOI at end of Write .... Yes
System Controller ..... Yes
Assert REN when SC ..... No
Enable Auto Serial Polling .. Yes
Enable CIC Protocol ..... No
Bus Timing 500nsec
Cable Length for High Speed . 15
Parallel Poll Duration ..... Default
Use this GPIB interface ..... Yes
Base I/O Address 02C0h
Interrupt Level 11
DMA Channel 5
DMA Transfer Mode ..... Demand
Select the GPIB board's base I/O address by entering the address using hex digits.
The valid range is 0000h to 03E0h in increments of 20h bytes.
A9 A8 A7 A6 A5
0
1 2 3 4 5
1
Push down the highlighted side.
F1: Help F6: Reset Value F9/Esc: Return to Map Ctl PgUp/PgDn: Next/Prev Board

Figure 8-2. Lower Level of wibconf

You can access the lower-level screens from the map level of `wibconf` by selecting a board or device and pressing <F8>. You can use the <Up>, <Down>, <PageUp>, and <PageDown> cursor keys to select a characteristic. For your convenience, cursor control keys and function keys are defined at the bottom of your computer screen.

Selecting the configuration settings for each device and board customizes the communications and other options to be used with that board or device. The access board uses these settings either when device functions are called to program the device or when board functions are called to program the board.

The following options are available at the lower level.

- Change Characteristics
- Change Board or Device
- Help
- Reset Value
- Return to Map

Change Characteristics To change a specific characteristic of a device or a board, move the cursor to, or click the mouse, on the field for that characteristic. You can also use <PageUp>, <PageDown>, <Home>, or <End> to select other characteristics of a device or board. When the cursor is on the characteristic, either use the left/right arrow keys to select between different options or input the option directly from the keyboard. Instructions on the right side of the screen inform you which method is appropriate for the selected characteristic.

Change Board or Device Use <Control-PageUp> and <Control-PageDown> to change to the next or previous GPIB board or device. For example, if you are editing `dev3` and press <Control-PageUp>, you will then be editing `dev4`.

Help Use the function key <F1> to access the comprehensive, online help feature of `wibconf`. The help information describes the functions and common terms associated with the lower level of `wibconf`.

Reset Value Use the function key <F6> to reset a characteristic option to its previous value.

Return to Map At the lower level, the function key <F9> or <Escape> returns you to the upper-level device map of `wibconf`.

Board and Device Configuration Options

To view detailed information about each characteristic, position the cursor in the field for that characteristic. For information on characteristics specific to a particular driver, check the getting started manual that came with your interface board. The following paragraphs describe the options available in `wibconf` for the NI-488.2 software for Windows.

Primary GPIB Address

All devices and boards must be assigned a unique primary address in the range hex 00 to hex 1E (0 to 30 decimal). The default primary address of all GPIB boards is 0.

The GPIB primary address of any device is set within the device, either with hardware switches or a by software program. The address set within the device must correspond to the address in the memory-resident driver. In the NI-488.2 driver for Windows, the default primary addresses of `dev1` through `dev16`, and `dev17` through `dev32`, are 1 through 16, respectively. Refer to the device documentation for instructions about setting the device address. GPIB boards do *not* have hardware switches to select the GPIB address.

The primary GPIB address is used to compute the talk and listen addresses of devices and boards. The NI-488.2 driver automatically forms a listen address by adding hex 20 to the primary address and a talk address by adding hex 40 to the primary address. For example, a primary address of hex 10 has a listen address of hex 30 and a talk address of hex 50.

Secondary GPIB Address

Any device or board using extended addressing must be assigned a secondary address in the range hex 60 to hex 7E (96 to 126 decimal), or you can select `NONE` to disable secondary addressing.

As with primary addressing, the secondary GPIB address of a device is set within that device, either with hardware switches or by a software program. The address set within the device must correspond to the address in the memory-resident driver. Refer to the device documentation for instructions about secondary addressing. The default setting for this characteristic is `NONE` for all boards and devices.

Timeout Setting

The timeout value is the approximate length of time that GPIB functions wait for data to be transferred or commands to be sent. It is also the length of time that the `ibwait` function waits for an event before returning, if the TIMO bit is set in the event mask. For example, if the SRQI bit and TIMO bit in the event mask are passed to the `ibwait` function and no SRQ is detected, then the function times out after the timeout period has elapsed. The default option for this characteristic is 10 s.

Serial Poll Timeouts (Device Characteristic Only)

This timeout value controls the length of time the driver waits for a serial poll response from a device. The IEEE 488.1 specification does not specify the length of time a Controller should wait for the response byte. The default of 1 s should work for most devices. If you seem to have problems with serial polls, try using a longer timeout value.

Terminate Read on EOS

Some devices send an EOS byte signaling the last byte of a data message. A `yes` response in this field causes the GPIB board to terminate a read operation when it receives the EOS byte. The default option for this characteristic is `no`.

Set EOI with EOS on Writes

A `yes` response in this field causes the GPIB board to assert the EOI line when the EOS byte is detected on a write operation. The default option for this characteristic is `no`.

Type of Compare on EOS

This field specifies the type of comparison to be made with the EOS byte. You can state whether to compare all eight bits or just the seven least significant bits, which are in ASCII or ISO (International Standards Organization) format. The default option for this characteristic is `7-bit`.

Note: ***This field is only meaningful if a*** `yes` ***response was given for either the*** `Set EOI with EOS on Write` ***field or the*** `Terminate Read on EOS` ***field.***

EOS Byte

Some devices can be programmed to terminate a read operation when a selected character is detected. A linefeed character (hex 0A) is a common EOS byte. The default option for this characteristic is 00H.

Note: ***The driver does not automatically append an EOS byte to the end of data strings on write operations. You must explicitly include this byte in your data string. The EOS byte is used only so that the driver terminates read operations properly.***

Send EOI at End of Write

Some devices, as Listeners, require that the Talker terminate a data message by asserting the EOI line with the last byte. A `yes` response causes the GPIB board to assert the EOI line on the last data byte. The default option for this characteristic is `yes`.

System Controller (Board Characteristic Only)

This field appears on the board characteristics screen only. The System Controller in a GPIB system is the device that maintains ultimate control over the bus. In some situations, such as a network of computers linked by the GPIB, another device may be System Controller and the GPIB board should be designated as *not* System Controller. A `no` response would designate *not* System Controller and a `yes` response would designate System Controller capability. The GPIB board is usually designated as System Controller. The default option for this characteristic is `yes`.

Note: ***You should not have more than one designated System Controller in any GPIB system***.

Assert REN when SC (Board Characteristic Only)

A `yes` response to this field causes Remote Enable (REN) to be asserted automatically whenever the board is placed online, if that the board has System Controller capability. If you give a `no` response, an explicit call to `ibsre` is required to assert REN. The default option for this characteristic is `no`.

Enable Auto Serial Polling (Board Characteristic Only)

This option enables or disables automatic serial polls of devices when the GPIB Service Request (SRQ) line is asserted. Positive poll responses are stored following the polls and can be read with the `ibrsp` device function. Normally, this feature does not conflict with devices that conform to the IEEE 488.1 specification. If a conflict exists with a device, use a `no` response for this field to disable this feature. The default option for this characteristic is `yes`.

Enable CIC Protocol (Board Characteristic Only)

If a device-level NI-488 function is called after control has been passed to another device, enabling this protocol causes the board to assert SRQ with a Serial Poll status byte of hex 42. The current Controller must recognize that the board wants to regain control. If the current Controller passes control back to the board, the device call proceeds as intended. If control is not passed within the timeout period, or if the CIC protocol is disabled, the ECIC error is returned. The default option for this characteristic is `no`.

Bus Timing (Board Characteristic Only)

This field specifies the T1 delay of the source handshake capability for the board. This delay determines the minimum amount of time, after the data is placed on the bus, that the board may assert DAV during a write or command operation. If the total length of the GPIB cable length in the system is less than 15 m, the value of 350 ns is appropriate.

Other factors might affect the choice of the T1 delay, although they are unlikely to affect your system setup. Refer to the ANSI/IEEE Standard 488.1-1987, Section 5.2, for more information about these other factors. The default for this option is 500 ns.

Cable Length for High Speed (Board Characteristic Only)

This field specifies the number of meters of GPIB cable you have in your system. If you use the HS488 high-speed protocol to communicate with HS488-compliant devices, you must specify the total number of meters of GPIB cable in your system. The System Controller, when it initializes the GPIB, must send this information to all HS488 devices so that high-speed transfers occur without errors.

Parallel Poll Duration (Board Characteristic Only)

This field specifies the length of time the driver waits when conducting a parallel poll. For a normal bus configuration (the Controller and devices on the same bus) use the default duration of 2 µs. If you are using a GPIB bus extender in transparent parallel poll mode, you should increase the poll duration to 10 µs so that the bus extender can operate transparently to your applications.

Use This GPIB Interface (Board Characteristic Only)

If you do not want the driver to try to access a board at the selected base address (because you do not have a board in the system), select `no` for this option. When this field is set to `no`, the driver returns the EDVR error as soon as a program tries to access the board. By default, access board `gpib0` is enabled, and `gpib1`, `gpib2`, and `gpib3` are disabled.

Base I/O Address (Board Description Only)

This field specifies the I/O address of the GPIB board. It must be set to the same value as the base I/O address setting for the GPIB board itself. Setting the base I/O address level is explained in the getting started manual that you received with your GPIB interface board.

Note: ***On some systems, this field is read-only and you cannot change it using*** `wibconf`. ***Refer to your getting started manual for information on how to change the I/O address.***

DMA Channel (Board Characteristic Only)

This field specifies the DMA channel used by the GPIB board. It must be set to the same value as the DMA channel (arbitration level for Micro Channel systems) setting for the GPIB board itself. Setting the DMA channel is explained in the getting started manual that you received with your GPIB interface board.

Note: ***On some systems, you can only enable or disable the use of DMA with the*** `wibconf` ***utility. You cannot change the DMA channel setting. Refer to your getting started manual for information on how to change the DMA channel.***

Interrupt Jumper Setting (Board Characteristic Only)

This field specifies the interrupt line used by the GPIB board. It must be set to the same value as the interrupt level setting for the GPIB board itself. Setting the interrupt level is explained in the getting started manual that you received with your GPIB board.

Note: ***On some systems, you can only enable or disable the use of interrupts with the*** `wibconf` ***utility. You cannot change the IRQ request line. Refer to your getting started manual for information on how to change the interrupt level.***

Enable Repeat Addressing (Device Characteristic Only)

Normally, devices are not addressed each time a read or write operation is performed. If `no` is selected, read or write operations do not readdress the selected device if the same operation was just performed with that device. Avoiding readdressing saves some time when you have several GPIB operations to perform. But it might be a problem for some older IEEE 488.1 devices that require their GPIB address to be sent with each I/O operation. Select `yes` to enable repeat addressing in such a situation. The default option for this characteristic is `no`.

GPIB-PCII/IIA Mode Switch

The driver that is included with the GPIB-PCII and GPIB-PCIIA interface board kits is the same for each kit and can run with both boards. Use this field to indicate the type of board that is installed in your system. You can have both a GPIB-PCII and a GPIB-PCIIA interface board installed in your system at the same time.

Default Configurations in wibconf

This section lists the default configuration values of the NI-488.2 driver.

- Thirty-two devices with symbolic names `dev1` through `dev32`.
- Four access boards with symbolic names `gpib0`, `gpib1`, `gpib2`, and `gpib3`. You cannot change the access board names.
- Access board `gpib0` is enabled. `gpib1`, `gpib2`, and `gpib3` are disabled.
- The GPIB addresses of the first 16 devices are the same as the device number. For example, `dev` is at address 1. These 16 devices are assigned to the access board `gpib0`.
- The remaining 16 devices (that is, devices 17 through 32) are assigned to the access board `gpib1`. Their GPIB addresses range from 1 through 16, respectively. For example, `dev17` is at address 1.
- Each GPIB interface board is System Controller for its independent bus and has a GPIB address of 0.
- The END message is sent with the last byte of each data message to a device. No end-of-string (EOS) character is recognized.
- The timeout value for I/O and wait functions is set for 10 s.
- Each GPIB board and device is set to perform I/O transfers using DMA.
- Automatic serial polling is enabled.
- At the end of each NI-488.2 routine, the NI-488.2 driver leaves the bus in its currently addressed state (IEEE 488.2 standard).

Exiting wibconf

After you have made all your changes, you can exit `wibconf` by pressing <F9> or <Esc>. The program prompts you to save the changes. Select `Yes` to save the changes, `No` to discard the changes, or `cancel` to remain in `wibconf`.

Checking for Errors

Unless you started in expert mode, `wibconf` checks for possible problems before quitting. It alerts you to situations such as the following:

- GPIB addressing conflict between a device and its access board
- GPIB boards not present in the host machine at the specified address
- Timeouts disabled on a device or board

To disable automatic checking when starting `wibconf`, use the `-e` option.

wibconf Batch Mode

The `wibconf` batch mode offers an alternative method for modifying the configurations of the NI-488.2 driver. Batch mode can also modify the loaded driver.

In batch mode, the configuration information is contained in a configuration file that you create. To use the configuration file that you created, enter the following command:

`wibconf -b` *`filename`*

where *`filename`* is the name of the configuration file. You must include at least one space between `-b` and *`filename`*.

The configuration file is a free-form text file consisting of pairs of items. Each item must be separated by at least one space or new line character. The first item of a pair is a mnemonic that represents a board or device characteristic or a device map configuration function (for example, `rename`, `connect`, `disconnect`) to be configured. The second item of a pair is the value to which the first item (mnemonic) should be set.

Before configuring a board or device, that board or device must be found by the `find name#` pair, where `name` is `board` or `device` and `#` is the index of the GPIB board or device you are configuring.

When `wibconf` is run in batch mode, it checks the syntax of the first item in the pair and the value range of the second item, and it reports any errors as it finds them. If a value range error is found, the correct range of values is displayed. If any errors are found, the driver is not configured.

The following example is a sample configuration file followed by an explanation of each entry.

```
find device1 name plotter
find device2 disconnect
find device3 connect board1
find board0 pad 2 tmo 30sec eos 0x1E sc no
            bin 8-bit tmng 350nsec
```

This example makes the following changes:

1. Change the name of the first device to `plotter`.
2. Disconnect the second device.
3. Connect the third device to board 1 (`gpib1`).
4. Change the configuration of board 0 (`gpib0`).
   - Primary address to 2 (`pad 2`)
   - Timeout setting to 30 s (`tmo 30sec`)
   - EOS byte to hex 1E (`eos 0x1E`)
   - System Controller capability to NO (`sc no`)
   - Type of compare on EOS to 8-bit (`bin 8-bit`)
   - GPIB bus timing to 350 ns (`tmng 350nsec`)

Table 8-2 contains a list of the valid pairs of items. You can enter number values as decimal or hexadecimal. Hexadecimal numbers must be preceded with `0x`. For example, decimal 64 would be written as `0x40`. For more information about valid input values, refer to the relevant function descriptions in the *NI-488.2 Function Reference Manual for DOS/Windows*.

Table 8-2. wibconf Batch Mode Command Pairs

First Item		Second Item
Mnemonic	**Explanation**	
find	Find board or device	board# or device#
pad	Primary GPIB address	number
sad	Secondary GPIB address	number
tmo	Timeout settings	mnemonic
xeos	Set EOI with EOS on write	yes or no
bin	Type of compare on EOS	7-bit or 8-bit
eot	Set EOI with last byte of write	yes or no
sc	System Controller (board only)	yes or no
sre	Assert REN when SC (board only)	yes or no
spoll	Enable auto serial polling (board only)	yes or no
tmng	Timing (board only)	2msec, 500nsec, or 350nsec
cic_prot	CIC protocol (board only)	yes or no
int	Interrupt setting (board only)	number
port	Base I/O address (board only)	number
dma	DMA channel (board only)	number
raddr	Repeat addressing (device only)	yes or no
name	Rename a device (device only)	device name
connect	Connect a device to a board (device only)	board#
disconnect	Disconnect a device from a board (device only)	no value
type	Switch the current board to PCII or PCIIA mode	PCII or PCIIA
pplength	Parallel poll duration (board only)	mnemonic
useboard	Use this interface board (board only)	yes or no
spolltmo	Serial Poll Timeout (device only)	mnemonic

Appendix A
Status Word Conditions

This appendix gives a detailed description of the conditions reported in the status word, `ibsta`.

For information about how to use `ibsta` in your application program, refer to Chapter 3, *Developing Your Application.*

If a function call returns an ENEB or EDVR error, all status word bits except the ERR bit are cleared, indicating that it is not possible to obtain the status of the GPIB board.

Each bit in `ibsta` can be set for device calls (dev), board calls (brd), or both (dev, brd).

The following table lists the status word bits.

Table A-1. Status Word Bits

Mnemonic	Bit Pos.	Hex Value	Type	Description
ERR	15	8000	dev, brd	GPIB error
TIMO	14	4000	dev, brd	Time limit exceeded
END	13	2000	dev, brd	END or EOS detected
SRQI	12	1000	brd	SRQ interrupt received
RQS	11	800	dev	Device requesting service
SPOLL	10	400	brd	Board has been serial polled by Controller
EVENT	9	200	brd	DCAS, DTAS, or IFC event has occurred
CMPL	8	100	dev, brd	I/O completed
LOK	7	80	brd	Lockout State
REM	6	40	brd	Remote State
CIC	5	20	brd	Controller-In-Charge
ATN	4	10	brd	Attention is asserted
TACS	3	8	brd	Talker
LACS	2	4	brd	Listener
DTAS	1	2	brd	Device Trigger State
DCAS	0	1	brd	Device Clear State

ERR (dev, brd)

ERR is set in the status word following any call that results in an error. You can determine the particular error by examining the error variable `iberr`. Appendix B, *Error Codes and Solutions*, describes error codes that are recorded in `iberr` along with possible solutions. ERR is cleared following any call that does not result in an error.

TIMO (dev, brd)

TIMO indicates that the timeout period has been exceeded. TIMO is set in the status word following an `ibwait` call if the TIMO bit of the `ibwait` mask parameter is set and the time limit expires. TIMO is also set following any synchronous I/O functions (for example, `ibcmd`, `ibrd`, `ibwrt`, `Receive`, `Send`, and `SendCmds`) if a timeout occurs during one of these calls. TIMO is cleared in all other circumstances.

END (dev, brd)

END indicates either that the GPIB EOI line has been asserted or that the EOS byte has been received, if the software is configured to terminate a read on an EOS byte. If the GPIB board is performing a shadow handshake as a result of the `ibgts` function, any other function can return a status word with the END bit set if the END condition occurs before or during that call. END is cleared when any I/O operation is initiated.

Some applications might need to know the exact I/O read termination mode of a read operation–EOI by itself, the EOS character by itself, or EOI plus the EOS character. You can use the `ibconfig` function (option `IbcEndBitIsNormal`) to enable a mode in which the END bit is set only when EOI is asserted. In this mode, if the I/O operation completes because of the EOS character by itself, END is not set. The application should check the last byte of the received buffer to see if it is the EOS character.

SRQI (brd)

SRQI indicates that a GPIB device is requesting service. SRQI is set whenever the GPIB board is CIC, the GPIB SRQ line is asserted, and the automatic serial poll capability is disabled. SRQI is cleared either when the GPIB board ceases to be the CIC or when the GPIB SRQ line is unasserted.

RQS (dev)

RQS appears in the status word only after a device-level call. It indicates that one or more serial poll response bytes are waiting in the device's serial poll response queue. Automatic serial poll responses are not stored in the response queue unless they have bit 6 set.

An automatic serial poll occurs either as a result of a call to `ibwait`, or automatically, if automatic serial polling is enabled. If the serial poll response queue is not empty, `ibrsp` returns the oldest byte stored in the queue. To empty the response queue, call `ibrsp` repeatedly until RQS is no longer set in the device's status word.

SPOLL (brd)

Use SPOLL in Talker/Listener applications to determine when the Controller has serial polled the GPIB board. The SPOLL bit is disabled by default. Use the `ibconfig` function (option `IbcSPollBit`) to enable it. When this bit is enabled, it is set after the board has been polled. SPOLL is cleared on any call immediately after an `ibwait` call, if the SPOLL bit was set in the wait mask, or immediately following a call to `ibrsv`.

EVENT (brd)

Use EVENT in Talker/Listener applications (applications in which the GPIB interface is not the Controller) to monitor the order of GPIB device clear, group execute trigger, and send interface clear commands. The usual DCAS and DTAS bits of `ibsta` might be insufficient.

The EVENT bit is disabled by default. If you want to use this bit, you must use the `ibconfig` function (option `IbcEventQueue`) to enable it. When you enable this bit, the DCAS and DTAS bits are disabled. When an event occurs, this bit is set and any I/O in progress is aborted. The application can then call the `ibevent` function to determine which event occurred.

CMPL (dev, brd)

CMPL indicates the condition of I/O operations. It is set whenever an I/O operation is complete. CMPL is cleared while the I/O operation is in progress.

LOK (brd)

LOK indicates whether the board is in a lockout state. While LOK is set, the `EnableLocal` routine or `ibloc` function is inoperative for that board. LOK is set whenever the GPIB board detects that the Local Lockout (LLO) message has been sent either by the GPIB board or by another Controller. LOK is cleared when the System Controller unasserts the Remote Enable (REN) GPIB line.

REM (brd)

REM indicates whether or not the board is in the remote state. REM is set whenever the Remote Enable (REN) GPIB line is asserted and the GPIB board detects that its listen address has been sent either by the GPIB board or by another Controller. REM is cleared in the following situations:

- When REN becomes unasserted
- When the GPIB board as a Listener detects that the Go to Local (GTL) command has been sent either by the GPIB board or by another Controller
- When the `ibloc` function is called while the LOK bit is cleared in the status word

CIC (brd)

CIC indicates whether the GPIB board is the Controller-In-Charge. CIC is set when the `SendIFC` routine or `ibsic` function is executed either while the GPIB board is System Controller or when another Controller passes control to the GPIB board. CIC is cleared either when the GPIB board detects Interface Clear (IFC) from the System Controller or when the GPIB board passes control to another device.

ATN (brd)

ATN indicates the state of the GPIB Attention (ATN) line. ATN is set whenever the GPIB ATN line is asserted, and it is cleared when the ATN line is unasserted.

TACS (brd)

TACS indicates whether the GPIB board is addressed as a Talker. TACS is set whenever the GPIB board detects that its talk address (and secondary address, if enabled) has been sent either by the GPIB board itself or by another Controller. TACS is cleared whenever the GPIB board detects the Untalk (UNT) command, its own listen address, a talk address other than its own talk address, or Interface Clear (IFC).

LACS (brd)

LACS indicates whether the GPIB board is addressed as a Listener. LACS is set whenever the GPIB board detects that its listen address (and secondary address, if enabled) has been sent either by the GPIB board itself or by another Controller. LACS is also set whenever the GPIB board shadow handshakes as a result of the `ibgts` function. LACS is cleared whenever the GPIB board detects the Unlisten (UNL) command, its own talk address, Interface Clear (IFC), or that the `ibgts` function has been called without shadow handshake.

DTAS (brd)

DTAS indicates whether the GPIB board has detected a device trigger command. DTAS is set whenever the GPIB board, as a Listener, detects that the Group Execute Trigger (GET) command has been sent by another Controller. DTAS is cleared on any call immediately following an `ibwait` call, if the DTAS bit is set in the `ibwait` mask parameter.

DCAS (brd)

DCAS indicates whether the GPIB board has detected a device clear command. DCAS is set whenever the GPIB board detects that the Device Clear (DCL) command has been sent by another Controller, or whenever the GPIB board as a Listener detects that the Selected Device Clear (SDC) command has been sent by another Controller. DCAS is cleared on any call immediately following an `ibwait` call, if the DCAS bit was set in the `ibwait` mask parameter. It also occurs on any call immediately following a read or write.

Appendix B
Error Codes and Solutions

This appendix lists a description of each error, some conditions under which it might occur, and possible solutions.

The following table lists the GPIB error codes.

Table B-1. GPIB Error Codes

Error Mnemonic	iberr Value	Meaning
EDVR	0	Windows error
ECIC	1	Function requires GPIB board to be CIC
ENOL	2	No Listeners on the GPIB
EADR	3	GPIB board not addressed correctly
EARG	4	Invalid argument to function call
ESAC	5	GPIB board not System Controller as required
EABO	6	I/O operation aborted (timeout)
ENEB	7	Nonexistent GPIB board
EDMA	8	DMA error
EOIP	10	Asynchronous I/O in progress
ECAP	11	No capability for operation
EFSO	12	File system error
EBUS	14	GPIB bus error
ESTB	15	Serial poll status byte queue overflow
ESRQ	16	SRQ stuck in ON position
ETAB	20	Table problem

EDVR (0)

EDVR is returned in the following cases:

- The board or device name passed to `ibfind` is not configured in the software. In this case, the variable `ibcntl` contains the DOS error code 2, *Device Not Found.*

- An invalid unit descriptor is passed to any function call. In this case, the variable `ibcntl` contains the DOS error code 6, *Invalid handle*.
- The driver (`gpib.dll`) is not installed.
- The driver configuration file `gpib.ini` is not located in the windows directory. In this case, the variable `ibcntl` contains the value -1.
- The driver file `gpib.ini` is in the windows directory but not compatible with the driver file `gpib.dll` that you are using. In this case, the variable `ibcntl` contains a negative value other than -1.

Solutions

- Use `ibdev` to open a device without specifying its symbolic name.
- Use only device or board names that are configured in the utility program `wibconf` as parameters to the `ibfind` function.
- Use the unit descriptor returned from `ibfind` as the first parameter in subsequent NI-488 functions. Examine the variable before the failing function to make sure the function has not been corrupted.
- Make sure the NI-488.2 driver is installed by checking that `gpib.dll` and `gpib.ini` are in the windows directory (usually `c:\windows`).

ECIC (1)

ECIC is returned when one of the following board functions or routines is called while the board is not CIC:

- Any device-level NI-488 functions that affect the GPIB
- Any board-level NI-488 functions that issue GPIB command bytes, such as `ibcmd`, `ibcmda`, `ibln`, `ibrpp`
- `ibcac`, `ibgts`
- Any of the NI-488.2 routines that issue GPIB command bytes, such as `SendCmds`, `PPoll`, `Send`, `Receive`

Solutions

- Use `ibsic` or `SendIFC` to make the GPIB board become CIC on the GPIB.
- Use `ibrsc 1` to make sure your GPIB board is configured as System Controller.
- In multiple CIC situations, always be certain that the CIC bit appears in the status word `ibsta` before attempting these calls. If it does not appear, you can perform an `ibwait` (for CIC) call to delay further processing until control is passed to the board.

ENOL (2)

ENOL usually occurs when a write operation is attempted with no Listeners addressed. For a device write, this error indicates that the GPIB address configured for that device in the software does not match the GPIB address of any device connected to the bus, that the GPIB cable is not connected to the device, or that the device is not powered on.

ENOL can also occur in situations in which the GPIB board is not the CIC and the Controller asserts ATN before the write call in progress has ended.

Solutions

- Make sure that the GPIB address of your device matches the GPIB address of the device to which you want to write data.
- Use the appropriate hex code in `ibcmd` to address your device.
- Check your cable connections and make sure at least two-thirds of your devices are powered on.
- Call `ibpad` (or `ibsad`, if necessary) to match the configured address to the device switch settings.
- Reduce the write byte count to that which is expected by the Controller.

EADR (3)

EADR occurs when the GPIB board is CIC and is not properly addressing itself before read and write functions. This error is usually associated with board-level functions.

EADR is also returned by the function `ibgts` when the shadow-handshake feature is requested and the GPIB ATN line is already unasserted. In this case, the shadow handshake is not possible and the error is returned to notify you of that fact.

Solutions

- Make sure that the GPIB board is addressed correctly before calling `ibrd`, `ibwrt`, `RcvRespMsg`, or `SendDataBytes`.
- Avoid calling `ibgts` except immediately after an `ibcmd` call. `ibcmd` causes ATN to be asserted.

EARG (4)

EARG results when an invalid argument is passed to a function call. The following are some examples:

- `ibtmo` called with a value not in the range 0 through 17
- `ibpad` or `ibsad` called with invalid addresses
- `ibppc` called with invalid parallel poll configurations
- A board-level NI-488 call made with a valid device descriptor, or a device-level NI-488 call made with a board descriptor.
- An NI-488.2 routine called with an invalid address.
- `PPollConfig` called with an invalid data line or sense bit.

Solutions

- Make sure that the parameters passed to the NI-488 function or NI-488.2 routine are valid.
- Do not use a device descriptor in a board function or vice-versa.

ESAC (5)

ESAC results when `ibsic`, `ibsre`, `SendIFC`, or `EnableRemote` is called when the GPIB board does not have System Controller capability.

Solutions

Give the GPIB board System Controller capability by calling `ibrsc 1` or by using `wibconf` to configure that capability into the software.

EABO (6)

EABO indicates that an I/O operation has been canceled, usually due to a timeout condition. Other causes for this error are calling `ibstop` or receiving the Device Clear message from the CIC while performing an I/O operation.

Frequently, the I/O is not progressing (the Listener is not continuing to handshake or the Talker has stopped talking), or the byte count in the call which timed out was more than the other device was expecting.

Solutions

- Use the correct byte count in input functions or have the Talker use the END message to signify the end of the transfer.
- Lengthen the timeout period for the I/O operation using `ibtmo`.
- Make sure that you have configured your device to send data before you request data.

ENEB (7)

ENEB occurs when no GPIB board exists at the I/O address specified in the configuration program. This happens when the board is not physically plugged into the system, the I/O address specified during configuration does not match the actual board setting, there is a system conflict with the base I/O address, or the `Use This GPIB Interface` field in `wibconf` is not set correctly.

Solutions

- Make sure there is a GPIB board in your computer that is properly configured both in hardware and software using a valid base I/O address.
- Make sure that the `Use This GPIB Interface` field in `wibconf` is set to `Yes`.

EDMA (8)

EDMA is returned when an error occurs using DMA for data transfers. If your computer has more than 16 MB of RAM and you do not have the National Instruments virtual GPIB device (`nivgpibd.386`) installed, the NI-488.2 software returns EDMA if you are using DMA and the data buffer is located in memory above 16 MB.

If you are using Windows 3.0 with DMA for data transfers, and you do not have the National Instruments virtual DMA device (`nivdmad.386`) installed, the NI-488.2 software returns EDMA if you try to use DMA to transfer data.

Solutions

- Install the appropriate virtual device in the `system.ini` file in the Windows directory in the `[386Enh]` section. The following line installs the virtual GPIB device:

  `device = drive:\path\nivgpib.386`

  where *drive* and *path* describe the location of `nivgpib.386` on your hard drive.

- By default, only one GPIB board at a time can perform DMA. If you need to perform DMA transfers on multiple GPIB boards at the same time, add a new section to your `system.ini` file, `[vgpibd]`. In this section add the option `NumBoardsUsingDMA` and set it equal to the number of boards that will be performing DMA. For example, if you want two boards to perform DMA concurrently, add the following lines to the bottom of your `system.ini` file:

```
[vgpibd]
NumBoardsUsingDMA=2
```

- To install the virtual DMA device, first change the default virtual DMA device line to a remark line by adding a semicolon. Then add a line to install the National Instruments virtual DMA device as follows:

```
;device = *vdmad
device = drive:\path\nivdmad.386
```

  where *drive* and *path* describe the location of `nivdmad.386` on your hard drive.

Note: ***You must restart Windows after you modify the*** `system.ini` ***file.***

- Alternatively, you can correct the EDMA problem by disabling DMA in the software. You can use `ibdma` to disable DMA.

EOIP (10)

EOIP occurs when an asynchronous I/O operation has not finished before some other call is made. During asynchronous I/O, you can only use `ibstop`, `ibwait`, and `ibonl` or perform other non-GPIB operations. Once the asynchronous I/O has begun, further GPIB calls other than `ibstop`, `ibwait`, or `ibonl` are strictly limited. If a call might interfere with the I/O operation in progress, the driver returns EOIP.

Solutions

Resynchronize the driver and the application before making any further GPIB calls. Resynchronization is accomplished by using one of the following three functions:

- `ibwait` If the returned `ibsta` contains CMPL then the driver and application are resynchronized.
- `ibstop` The I/O is canceled; the driver and application are resynchronized.
- `ibonl` The I/O is canceled and the interface is reset; the driver and application are resynchronized.

ECAP (11)

ECAP results when your GPIB board lacks the ability to carry out an operation or when a particular capability has been disabled in the software and a call is made that requires the capability.

Solutions

Check the validity of the call, or make sure your GPIB interface board and the driver both have the needed capability.

EFSO (12)

EFSO results when an `ibrdf` or `ibwrtf` call encounters a problem performing a file operation. Specifically, this error indicates that the function is unable to open, create, seek, write, or close the file being accessed. The specific DOS error code for this condition is contained in `ibcntl`.

Solutions

- Make sure the filename, path, and drive that you specified are correct.
- Make sure that the access mode of the file is correct.
- Make sure there is enough room on the disk to hold the file.

EBUS (14)

EBUS results when certain GPIB bus errors occur during device functions. All device functions send command bytes to perform addressing and other bus management. Devices are expected to accept these command bytes within the time limit specified by the default configuration or the `ibtmo` function. EBUS results if a timeout occurred while sending these command bytes.

Solutions

- Verify that the instrument is operating correctly.
- Check for loose or faulty cabling or several powered-off instruments on the GPIB.
- If the timeout period is too short for the driver to send command bytes, increase the timeout period.

ESTB (15)

ESTB is reported only by the `ibrsp` function. ESTB indicates that one or more serial poll status bytes received from automatic serial polls have been discarded because of a lack of storage space. Several older status bytes are available; however, the oldest is being returned by the `ibrsp` call.

Solutions

- Call `ibrsp` more frequently to empty the queue.
- Disable autopolling with the `ibconfig` function or the `wibconf` utility.

ESRQ (16)

ESRQ occurs only during the `ibwait` function or the `WaitSRQ` routine. ESRQ indicates that a wait for RQS is not possible because the GPIB SRQ line is stuck on. This situation can be caused by the following events:

- Usually, a device unknown to the software is asserting SRQ. Because the software does not know of this device, it can never serial poll the device and unassert SRQ.
- A GPIB bus tester or similar equipment might be forcing the SRQ line to be asserted.
- A cable problem might exist involving the SRQ line.

Although the occurrence of ESRQ warns you of a definite GPIB problem, it does not affect GPIB operations, except that you cannot depend on the RQS bit while the condition lasts.

Solutions

Check to see if other devices not used by your application are asserting SRQ. Disconnect them from the GPIB if necessary.

ETAB (20)

ETAB occurs only during the `FindLstn`, `FindRQS`, and `ibevent` functions. ETAB indicates that there was some problem with a table used by these functions.

- In the case of `FindLstn`, ETAB means that the given table did not have enough room to hold all the addresses of the Listeners found.
- In the case of `FindRQS`, ETAB means that none of the devices in the given table were requesting service.
- In the case of `ibevent`, ETAB means the event queue overflowed and event information was lost.

Solutions

In the case of `FindLstn`, increase the size of result arrays. In the case of `FindRQS`, check to see if other devices not used by your application are asserting SRQ. Disconnect them from the GPIB if necessary. In the case of ETAB returned from `ibevent`, call `ibevent` more often to empty the queue.

Appendix C
Customer Communication

For your convenience, this appendix contains forms to help you gather the information necessary to help us solve technical problems you might have as well as a form you can use to comment on the product documentation. Filling out a copy of the *Technical Support Form* before contacting National Instruments helps us help you better and faster.

National Instruments provides comprehensive technical assistance around the world. In the U.S. and Canada, applications engineers are available Monday through Friday from 8:00 a.m. to 6:00 p.m. (central time). In other countries, contact the nearest branch office. You may fax questions to us at any time.

Corporate Headquarters
(512) 795-8248
Technical support fax: (800) 328-2203
(512) 794-5678

Branch Offices	**Phone Number**	**Fax Number**
Australia	03 9 879 9422	03 9 879 9179
Austria	0662 45 79 90 0	0662 45 79 90 19
Belgium	02 757 00 20	02 757 03 11
Canada (Ontario)	519 622 9310	519 622 9311
Canada (Quebec)	514 694 8521	514 694 4399
Denmark	45 76 26 00	45 76 71 11
Finland	90 527 2321	90 502 2930
France	1 48 14 24 24	1 48 14 24 14
Germany	089 741 31 30	089 714 60 35
Hong Kong	2645 3186	2686 8505
Italy	02 48301892	02 48301915
Japan	03 5472 2970	03 5472 2977
Korea	02 596 7456	02 596 7455
Mexico	95 800 010 0793	5 520 3282
Netherlands	0348 433466	0348 430673
Norway	32 84 84 00	32 84 86 00
Singapore	2265886	2265887
Spain	91 640 0085	91 640 0533
Sweden	08 730 49 70	08 730 43 70
Switzerland	056 200 51 51	056 200 51 55
Taiwan	02 377 1200	02 737 4644
U.K.	01635 523545	01635 523154

Technical Support Form

Photocopy this form and update it each time you make changes to your software or hardware, and use the completed copy of this form as a reference for your current configuration. Completing this form accurately before contacting National Instruments for technical support helps our applications engineers answer your questions more efficiently.

If you are using any National Instruments hardware or software products related to this problem, include the configuration forms from their user manuals. Include additional pages if necessary.

Name _____

Company _____

Address _____

_____

Fax (___)_____ Phone (___)_____

Computer brand _____

Model _____ Processor _____

Operating system _____

Speed _____MHz RAM _____MB

Display adapter _____

Mouse _____yes _____no

Other adapters installed _____

Hard disk capacity _____MB Brand _____

Instruments used _____

National Instruments hardware product model _____

Revision _____

Configuration _____

(continues)

National Instruments software product _____

Version _____

Configuration _____

The problem is _____

_____

_____

_____

_____

_____

_____

_____

List any error messages _____

_____

_____

_____

_____

_____

_____

_____

The following steps will reproduce the problem _____

_____

_____

_____

_____

_____

_____

Documentation Comment Form

National Instruments encourages you to comment on the documentation supplied with our products. This information helps us provide quality products to meet your needs.

Title: **NI-488.2™ User Manual for Windows**

Edition Date: **January 1996**

Part Number: **370902A-01**

Please comment on the completeness, clarity, and organization of the manual.

(continues)

If you find errors in the manual, please record the page numbers and describe the errors.

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

Thank you for your help.

Name _____

Title _____

Company _____

Address _____

_____

Phone (_____) _____

Mail to: Technical Publications
National Instruments Corporation
6504 Bridge Point Parkway,
Austin, TX 78730-5039

Fax to: Technical Publications
National Instruments Corporation
(512) 794-5678

Glossary

Prefix	Meaning	Value
n-	nano-	10^{-9}
µ-	micro-	10^{-6}
m-	milli-	10^{-3}
k-	kilo-	10^{3}
M-	mega-	10^{6}

A

acceptor handshake — Listeners use this GPIB interface function to receive data, and all devices use it to receive commands. See *source handshake* and *handshake*.

access board — The GPIB board that controls and communicates with the devices on the bus that are attached to it.

ANSI — American National Standards Institute.

ASCII — American Standard Code for Information Interchange.

asynchronous — An action or event that occurs at an unpredictable time with respect to the execution of a program.

automatic serial polling (autopolling) — A feature of the NI-488.2 software in which serial polls are executed automatically by the driver whenever a device asserts the GPIB SRQ line.

B

base I/O address — See *I/O address*.

BIOS — Basic Input/Output System.

board-level function — A rudimentary function that performs a single operation.

C

CFE	Configuration Enable is the GPIB command which precedes CFGn and is used to place devices into their configuration mode.
CFGn	These GPIB commands (CFG1 through CFG15) follow CFE and are used to configure all devices for the number of meters of cable in the system so that HS488 transfers occur without errors.
CIC	See *Controller-In-Charge*.
Controller-In-Charge (CIC)	The device that manages the GPIB by sending interface messages to other devices.
CPU	Central processing unit.

D

DAV (Data Valid)	One of the three GPIB handshake lines. See *handshake*.
DCL	Device Clear is the GPIB command used to reset the device or internal functions of all devices. See *SDC*.
Device Clear	See DCL.
device-level function	A function that combines several rudimentary board operations into one function so that the user does not have to be concerned with bus management or other GPIB protocol matters.
DIO1 through DIO8	The GPIB lines that are used to transmit command or data bytes from one device to another.
DLL	Dynamic link library.
DMA (direct memory access)	High-speed data transfer between the GPIB board and memory that is not handled directly by the CPU. Not available on some systems. See *programmed I/O*.
driver	Device driver software installed within the operating system.

E

END or END message	A message that signals the end of a data string. END is sent by asserting the GPIB End or Identify (EOI) line with the last data byte.
EOI	A GPIB line that is used to signal either the last byte of a data message (END) or the parallel poll Identify (IDY) message.
EOS or EOS byte	A 7- or 8-bit end-of-string character that is sent as the last byte of a data message.
EOT	End of transmission.
ESB	The Event Status bit is part of the IEEE 488.2-defined status byte which is received from a device responding to a serial poll.

G

GET	Group Execute Trigger is the GPIB command used to trigger a device or internal function of an addressed Listener.
Go To Local	See *GTL*.
GPIB	General Purpose Interface Bus is the common name for the communications interface system defined in ANSI/IEEE Standard 488.1-1987 and ANSI/IEEE Standard 488.2-1987.
GPIB address	The address of a device on the GPIB, composed of a primary address (MLA and MTA) and an optional secondary address (MSA). The GPIB board has both a GPIB address and an I/O address.
GPIB board	Refers to the National Instruments family of GPIB interface boards.
Group Executed Trigger	See *GET*.
GTL	Go To Local is the GPIB command used to place an addressed Listener in local (front panel) control mode.

H

handshake	The mechanism used to transfer bytes from the Source Handshake function of one device to the Acceptor Handshake function of another device. The three GPIB lines DAV, NRFD, and NDAC are used in an interlocked fashion to signal the phases of the transfer, so that bytes can be sent asynchronously (for example, without a clock) at the speed of the slowest device. For more information about handshaking, refer to the ANSI/IEEE Standard 488.1-1987.
hex	Hexadecimal; a number represented in base 16, for example decimal 16 = hex 10.
high-level function	See *device-level function*.
Hz	Hertz.

I

`ibcnt`	After each NI-488.2 I/O function, this global variable contains the actual number of bytes transmitted.
`iberr`	A global variable that contains the specific error code associated with a function call that failed.
`ibsta`	At the end of each function call, this global variable (status word) contains status information.
IEEE	Institute of Electrical and Electronic Engineers.
interface message	A broadcast message sent from the Controller to all devices and used to manage the GPIB.
I/O (Input/Output)	In the context of this manual, the transmission of commands or messages between the computer via the GPIB board and other devices on the GPIB.
I/O address	The address of the GPIB board from the point of view of the CPU, as opposed to the GPIB address of the GPIB board. Also called port address or board address.
`ist`	An Individual Status bit of the status byte used in the Parallel Poll Configure function.

K

KB	Kilobytes.

L

LAD (Listen Address)	See *MLA*.
language interface	Code that enables an application program that uses NI-488 functions or NI-488.2 routines to access the driver.
listen address	See *MLA*.
Listener	A GPIB device that receives data messages from a Talker.
low-level function	See *board-level function*.

M

m	Meters.
MAV	The Message Available bit is part of the IEEE 488.2-defined status byte which is received from a device responding to a serial poll.
MB	Megabytes of memory.
memory-resident	Resident in RAM.
MLA (My Listen Address)	A GPIB command used to address a device to be a Listener. It can be any one of the 31 primary addresses.
MSA (My Secondary Address)	My Secondary Address is the GPIB command used to address a device to be a Listener or a Talker when extended (two byte) addressing is used. The complete address is a MLA or MTA address followed by an MSA address. There are 31 secondary addresses for a total of 961 distinct listen or talk addresses for devices.
MTA (My Talk Address)	A GPIB command used to address a device to be a Talker. It can be any one of the 31 primary addresses.
multitasking	The concurrent processing of more than one program or task.

N

NDAC (Not Data Accepted)	One of the three GPIB handshake lines. See *handshake*.
NRFD (Not Ready For Data)	One of the three GPIB handshake lines. See *handshake*.

P

parallel poll	The process of polling all configured devices at once and reading a composite poll response. See *serial poll*.
PIO	See *programmed I/O*.
PPC (Parallel Poll Configure)	Parallel Poll Configure is the GPIB command used to configure an addressed Listener to participate in polls.
PPD (Parallel Poll Disable)	Parallel Poll Disable is the GPIB command used to disable a configured device from participating in polls. There are 16 PPD commands.
PPE (Parallel Poll Enable)	Parallel Poll Enable is the GPIB command used to enable a configured device to participate in polls and to assign a DIO response line. There are 16 PPE commands.
PPU (Parallel Poll Unconfigure)	Parallel Poll Unconfigure is the GPIB command used to disable any device from participating in polls.
programmed I/O	Low-speed data transfer between the GPIB board and memory in which the CPU moves each data byte according to program instructions. See *DMA*.

R

RAM	Random-access memory.
resynchronize	The NI-488.2 software and the user application must resynchronize after asynchronous I/O operations have completed.
RQS	Request Service.

S

s	Seconds.
SDC	Selected Device Clear is the GPIB command used to reset internal or device functions of an addressed Listener. See *DCL* and *IFC*.
serial poll	The process of polling and reading the status byte of one device at a time. See *parallel poll*.
Service Request	See *SRQ*.
source handshake	The GPIB interface function that transmits data and commands. Talkers use this function to send data, and the Controller uses it to send commands. See *acceptor handshake* and *handshake*.
SPD (Serial Poll Disable)	Serial Poll Disable is the GPIB command used to cancel an SPE command.
SPE (Serial Poll Enable)	Serial Poll Enable is the GPIB command used to enable a specific device to be polled. That device must also be addressed to talk. See *SPD*.
SRQ (Service Request)	The GPIB line that a device asserts to notify the CIC that the device needs servicing.
status byte	The IEEE 488.2-defined data byte sent by a device when it is serially polled.
status word	See *ibsta*.
synchronous	Refers to the relationship between the NI-488.2 driver functions and a process when executing driver functions is predictable; the process is blocked until the driver completes the function.
System Controller	The single designated Controller that can assert control (become CIC of the GPIB) by sending the Interface Clear (IFC) message. Other devices can become CIC only by having control passed to them.

T

TAD (Talk Address)	See *MTA*.
Talker	A GPIB device that sends data messages to Listeners.
TCT	Take Control is the GPIB command used to pass control of the bus from the current Controller to an addressed Talker.
timeout	A feature of the NI-488.2 driver that prevents I/O functions from hanging indefinitely when there is a problem on the GPIB.
TLC	An integrated circuit that implements most of the GPIB Talker, Listener, and Controller functions in hardware.

U

`ud` (unit descriptor)	A variable name and first argument of each function call that contains the unit descriptor of the GPIB interface board or other GPIB device that is the object of the function.
UNL	Unlisten is the GPIB command used to unaddress any active Listeners.
UNT	Untalk is the GPIB command used to unaddress an active Talker.

W

`wibconf`	The NI-488.2 driver configuration program for Windows.
`wibic`	The Interface Bus Interactive Control program for Windows is used to communicate with GPIB devices, troubleshoot problems, and develop your application.

Index

Symbols

! (repeat previous function) function, wibic, 5-15
$ (execute indirect file) function, wibic, 5-16
+ (turn display on) function, wibic, 5-15
- (turn display off) function, wibic, 5-15

A

Active Controller. *See* Controller-in-Charge (CIC).
addresses. *See* GPIB addresses.
AllSpoll routine, 7-8, 7-9 to 7-10
ANSI/IEEE Standard 488.1-1987. *See* GPIB.
application development. *See also* debugging.
    accessing NI-488.2 DLL, 3-1
    application examples
        asynchronous I/O, 2-6 to 2-7
        basic communication, 2-2 to 2-3
        basic communication with IEEE 488.2-compliant devices, 2-14 to 2-15
        clearing and triggering devices, 2-4 to 2-5
        end-of-string mode, 2-8 to 2-9
        non-controller example, 2-20 to 2-21
        parallel polls, 2-18 to 2-19
        serial polls using NI-488.2 routines, 2-16 to 2-17
        service requests, 2-10 to 2-13
        source code files, 2-1
    choosing between NI-488 functions and NI-488.2 routines, 3-1 to 3-3
    compiling, linking, and running applications
        Borland C++, 3-17
        direct entry with C, 3-22
        direct entry with Visual Basic, 3-23
        Microsoft C, 3-17
        Visual Basic applications, 3-18
    global variables for checking status, 3-3 to 3-5
        count variables - ibcnt and ibcntl, 3-5
        error variable - iberr, 3-5
        status word - ibsta, 3-3 to 3-4
    NI-488 applications
        clearing devices, 3-8
        configuring devices, 3-9
        flowchart of programming with device-level functions, 3-7
        general steps and examples, 3-8 to 3-10
        items to include, 3-6

opening devices, 3-8
placing device offline, 3-10
processing of data, 3-10
program shell (illustration), 3-7
reading measurement, 3-10
triggering devices, 3-9
waiting for measurement, 3-9 to 3-10
NI-488 functions, 3-2 to 3-3
advantages, 3-2
board functions, 3-2 to 3-3
device functions, 3-2
one device per board, 3-2 to 3-3
NI-488.2 applications
configuring instruments, 3-15
finding all Listeners, 3-13
flowchart of programming with routines, 3-12
general steps and examples, 3-13 to 3-16
identifying instruments, 3-13 to 3-14
initialization, 3-13
initializing instruments, 3-14
items to include, 3-11
placing board offline, 3-16
processing of data, 3-16
program shell (illustration), 3-12
reading measurement, 3-16
triggering instruments, 3-15
waiting for measurement, 3-15 to 3-16
NI-488.2 routines, 3-3
wibic for communicating with devices, 3-5
asynchronous I/O application example, 2-6 to 2-7
ATN (attention) line, 1-3
ATN status word condition
bit position, hex value, and type (table), 3-4
description, A-4
automatic serial polling. *See* serial polling.
auxiliary functions, wibic
! (repeat previous function), 5-15
$ (execute indirect file), 5-16
+ (turn display on), 5-15
- (turn display off), 5-15
buffer (set buffer display mode), 5-17
help (display help information), 5-15
n* (repeat function n times), 5-16
print (display the ASCII string), 5-17
set (select device or board), 5-15
table of functions, 5-14

B

base I/O address, setting, 8-10
batch mode for wibconf utility, 8-13 to 8-15
    command pairs (table), 8-15
    configuration file, 8-13 to 8-14
board configuration. *See* wibconf utility.
board functions. *See* NI-488 functions.
boards
    disabling access to GPIB board, 8-10
    testing with wibtest, 4-1 to 4-2
Borland C++. *See* C language.
buffer command (set buffer display mode), 5-17
buffer history, viewing, 6-3
bus timing, setting, 8-10

C

C language
    compiling, linking, and running applications, 3-17
    direct entry, 3-18 to 3-22
    files available with NI-488.2 software, 1-7
cables
    checking with wibtest, 4-2
    setting cable length for high-speed data transfers, 8-10
Call Details window, GPIB Spy utility, 6-2
calls
    highlighting, 6-3
    trapping, 6-3
CIC. *See* Controller-in-Charge (CIC).
CIC protocol
    enabling, 8-9
    making GPIB board Controller-in-Charge, 7-3 to 7-4
CIC status word condition
    bit position, hex value, and type (table), 3-4
    description, A-4
clearing and triggering devices, example, 2-4 to 2-5
CMPL status word condition
    bit position, hex value, and type (table), 3-4
    description, A-3
communication application examples
    basic communication, 2-2 to 2-3
        with IEEE 488.2-compliant devices, 2-14 to 2-15
communication errors, 4-6
    repeat addressing, 4-6
    termination method, 4-6

compiling, linking, and running applications
Borland C++, 3-17
direct entry with C, 3-18 to 3-22
direct entry with Visual Basic, 3-23
Microsoft C, 3-17
Visual Basic, 3-18
configuration, 1-4 to 1-6. *See also* GPIB Spy utility; wibconf utility.
controlling more than one board, 1-5
linear and star system configuration (illustration), 1-4
requirements, 1-5 to 1-6
configuration errors, 4-5 to 4-6
configuration file for wibconf batch mode, 8-13 to 8-14
Configure (CFGn) message, 7-2
Configure Enable (CFE) message, 7-2
Controller-in-Charge (CIC)
Active Controller as CIC, 1-1
CIC protocol, 7-3, 8-9
configuring GPIB board as CIC, 7-3 to 7-4, 8-9
System Controller as, 1-1
Controllers
definition, 1-1
emulation of non-controller GPIB, example, 2-20 to 2-21
idle Controller, 1-1
System Controller, 1-1, 8-9
count, in wibic, 5-10
count variables - ibcnt and ibcntl, 3-5
customer communication, *xvi*, C-1

D

data lines, 1-2
data transfers
high-speed (HS488), 7-2 to 7-3
enabling, 7-2 to 7-3
system configuration effects, 7-3
terminating, 7-1 to 7-2
DAV (data valid) line, 1-3
DCAS status word condition
bit position, hex value, and type (table), 3-4
description, A-5
monitoring for messages from Controller, 7-4
debugging. *See also* GPIB Spy utility.
common questions, 4-7 to 4-8 to 4-9
communication errors, 4-6
repeat addressing, 4-6
termination method, 4-6
configuration errors, 4-5 to 4-6

global status variables, 4-3
GPIB error codes (table), 4-4 to 4-5
GPIB Spy utility, 4-4
GPIBInfo utility, 4-2 to 4-3
timing errors, 4-6
wibic utility, 4-4
wibtest diagnostics, 4-1 to 4-2
cable connections, 4-2
presence of board, 4-1 to 4-2
presence of driver, 4-1
default configurations in wibconf, 8-12
DevClear routine, 3-14
device configuration. *See* wibconf utility.
device functions. *See* NI-488 functions.
device-level calls and bus management, 7-3 to 7-4
direct access to NI-488.2 dynamic link library
compiling, linking, and running applications
using C language, 3-18 to 3-22
using Visual Basic, 3-23
requirements, 3-1
DMA channel, setting, 8-11
documentation
conventions used in manual, *xv*
organization of manual, *xi*
related documentation, *xvi*
using the manual set, *xiii*
drivers
driver and driver utilities for NI-488.2 software, 1-6 to 1-7
reconfiguring, 4-5
testing with wibtest, 4-1
DTAS status word condition
bit position, hex value, and type (table), 3-4
description, A-5
monitoring for message from Controller, 7-4
dynamic link library, GPIB. *See* NI-488.2 DLL.

E

EABO error code
definition (table), 4-5
description, B-5
EADR error code
definition (table), 4-5
description, B-3 to B-4
EARG error code
definition (table), 4-5
description, B-4
wibic example, 5-12

EBUS error code
    definition (table), 4-5
    description, B-8
ECAP error code
    definition (table), 4-5
    description, B-7
ECIC error code
    definition (table), 4-5
    description, B-2 to B-3
EDMA error code
    definition (table), 4-5
    description, B-5 to B-6
EDVR error code
    definition (table), 4-5
    description, B-1 to B-2
    wibic example, 5-11
EFSO error code
    definition (table), 4-5
    description, B-7
end-of-string character. *See* EOS.
END status word condition
    bit position, hex value, and type (table), 3-4
    description, A-1 to A-2
ENEB error code
    definition (table), 4-5
    description, B-5
    wibic example, 5-11
ENOL error code
    definition (table), 4-5
    description, B-3
EOI (end or identify) line
    purpose (table), 1-3
    send EOI at end of write, 8-9
    set EOI with EOS on writes, 8-8
    termination of data transfers, 7-1 to 7-2
EOIP error code
    definition (table), 4-5
    description, B-6 to B-7
EOS
    configuring EOS mode, 7-1
    end-of-string mode application example, 2-8 to 2-9
    set EOI with EOS on writes, 8-8
    setting character for EOS byte, 8-8
    terminate read on EOS, 8-8
    type of compare on EOS, 8-8
ERR status word condition
    bit position, hex value, and type (table), 3-4
    description, A-1

error codes and solutions
    EABO, B-5
    EADR, B-3 to B-4
    EARG, B-4
    EBUS, B-8
    ECAP, B-7
    ECIC, B-2 to B-3
    EDMA, B-5 to B-6
    EDVR, B-1 to B-2
    EFSO, B-7
    ENEB, B-5
    ENOL, B-3
    EOIP, B-6 to B-7
    ESAC, B-4 to B-5
    ESRQ, B-8 to B-9
    ESTB, B-8
    ETAB, B-9
    table of codes, 4-5
error conditions
    checking for errors before exiting wibconf, 8-13
    communication errors, 4-6
        repeat addressing, 4-6
        termination method, 4-6
    configuration errors, 4-5 to 4-6
    timing errors, 4-6
    wibic error information, 5-9 to 5-10
error variable - iberr, 3-5
ESAC error code
    definition (table), 4-5
    description, B-4
ESRQ error code
    definition (table), 4-5
    description, B-8 to B-9
ESTB error code
    definition (table), 4-5
    description, B-8
ETAB error code
    definition (table), 4-5
    description, B-8
EVENT bit, enabling, 7-5
event queue, 7-5
Event Status bit (ESB), 7-6
EVENT status word condition
    bit position, hex value, and type (table), 3-4
    description, A-3
execute indirect file ($) function, wibic, 5-16

F

fax technical support, C-1
FindLstn routine, 3-13
FindRQS routine, 7-8, 7-9
functions. *See* auxiliary functions, wibic; NI-488 functions.

G

General Purpose Interface Bus. *See* GPIB.
global variables, 3-3 to 3-5
    count variables - ibcnt and ibcntl, 3-5
    debugging applications, 4-3
    error variable - iberr, 3-5
    status word - ibsta, 3-3 to 3-4, A-1 to A-5
GPIB
    configuration, 1-4 to 1-6. *See also* wibconf utility.
        controlling more than one board, 1-5
        linear and star system configuration (illustration), 1-4
        requirements, 1-5 to 1-6
    definition, 1-1
    overview, 1-1
    sending messages across, 1-2 to 1-3
        data lines, 1-2
        handshake lines, 1-3
        interface management lines, 1-3
    Talkers, Listeners, and Controllers, 1-1
GPIB addresses
    address bit configuration (illustration), 1-2
    enabling repeat addressing, 8-11
    listen address, 1-2
    primary, 1-2, 8-7
    purpose, 1-2
    repeat addressing, 4-6
    secondary, 1-2, 8-7
    syntax in wibic, 5-4 to 5-5
    talk address, 1-2
GPIB/PCII/IIA mode switch, 8-11
GPIB programming techniques
    device-level calls and bus management, 7-3 to 7-4
    high-speed data transfers, 7-2 to 7-3
        enabling HS488, 7-2
        system configuration effects, 7-3
    parallel polling, 7-10 to 7-12
        implementing, 7-10 to 7-12
        using NI-488 functions, 7-10 to 7-11
        using NI-488.2 routines, 7-12

serial polling, 7-5 to 7-10
automatic serial polling, 7-6 to 7-7
autopolling and interrupts, 7-7
stuck SRQ state, 7-6 to 7-7
service requests
from IEEE 488 devices, 7-5
from IEEE 488.2 devices, 7-6
SRQ and serial polling
with NI-488 device functions, 7-7 to 7-8
with NI-488.2 routines, 7-8 to 7-10
Talker/Listener applications, 7-4 to 7-5
event queue, 7-5
requesting service, 7-5
waiting for messages from Controller, 7-4
termination of data transfers, 7-1 to 7-2
waiting for GPIB conditions, 7-3
GPIB Spy utility. *See also* debugging.
configuration options
Buffer History, 6-3
Calls to Highlight, 6-3
Calls to Trap On, 6-3
Set Highlight Color, 6-3
Store Configuration, 6-4
debugging applications, 4-4
exiting, 6-2
Help menu, 6-5
main window (illustration), 6-1
output options
Clear Screen, 6-4
To File, 6-4
To Screen, 6-4
overview, 6-1
performance considerations, 6-5
purpose, 1-7
starting, 6-1
viewing call details, 6-2
gpib.dll file, 1-6. *See also* NI-488.2 DLL.
GPIBInfo utility
purpose, 1-7
running, 4-2 to 4-3
gpib.ini file, 1-6, B-2

H

handshake lines, 1-3
Help (display help information) function, 5-15

high-speed data transfers (HS488), 7-2 to 7-3
    enabling HS488, 7-2 to 7-3
    setting cable length, 8-10
    system configuration effects, 7-3
highlight color, setting, 6-3

I

ibask function, 7-3
ibclr function
    clearing devices, 3-8
    using in wibic, example, 5-2
ibcmd function, 7-2
ibcnt and ibcntl variables. *See* count variables - ibcnt and ibcntl.
ibconfig function
    configuring GPIB board as CIC, 7-3 to 7-4
    determining assertion of EOI line, 7-1
    enabling autopolling, 7-6
    enabling EVENT bit, 7-5
    enabling high-speed data transfers, 7-2 to 7-3
    reconfiguring GPIB driver, 4-5 to 4-6
ibdev function
    conducting parallel polls, 7-11
    opening devices, 3-8
    using in wibic, 5-10 to 5-11
        example, 5-2
ibdiag utility
    purpose, 1-7
    testing hardware configuration, 4-6
ibeos function, 7-1
ibeot function, 7-1
iberr. *See* error variable - iberr.
ibevent function, 7-5
ibfind function, 5-10
ibonl function
    placing board offline, 3-16
    placing device offline, 3-10
    using in wibic, example, 5-4
ibppc function
    conducting parallel polls, 7-10
    unconfiguring device for parallel polling, 7-12
ibrd function
    using in wibic, 5-12
        example, 5-4
ibrpp function, 7-11

ibrsp function
    conducting serial polls, 7-6, 7-7 to 7-8
    using in wibic, example, 5-3
ibsta. *See* status word - ibsta.
ibtrg function
    triggering devices, 3-9
    using in wibic, example, 5-3
ibwait function
    conducting serial polls, 7-7 to 7-8
    Talker/Listener applications, 7-4
    terminating stuck SRQ state, 7-6 to 7-7
    using in wibic, example, 5-3
    waiting for GPIB conditions, 7-3
    waiting for measurement, 3-9 to 3-10
ibwrt function
    configuring devices, 3-9
    using in wibic, 5-12
        example, 5-3
*IDN? query, 3-13 to 3-14
IEEE Standard 488.1-1987. *See* GPIB.
IFC (interface clear) line, 1-3
Interface Bus Interactive Control utility. *See* wibic utility.
interface management lines, 1-3
interrupt jumper, setting, 8-11
interrupts and autopolling, 7-6 to 7-7

L

LACS status word condition
    bit position, hex value, and type (table), 3-4
    description, A-4
    monitoring for message from Controller, 7-4
linking applications. *See* compiling, linking, and running applications.
listen address, setting, 1-2
Listeners, 1-1. *See also* Talker/Listener applications.
LOK status word condition
    bit position, hex value, and type (table), 3-4
    description, A-3

M

manual. *See* documentation.
Message Available (MAV) bit, 7-6

messages, sending across GPIB, 1-2 to 1-3
    data lines, 1-2
    handshake lines, 1-3
    interface management lines, 1-3
Microsoft C. *See* C language.

N

n* (repeat function n times) function, wibic, 5-16
NDAC (not data accepted) line, 1-3
NI-488 applications, programming
    clearing devices, 3-8
    configuring devices, 3-9
    flowchart of programming with device-level functions, 3-7
    general steps and examples, 3-8 to 3-10
    items to include, 3-6
    opening devices, 3-8
    placing device offline, 3-10
    processing of data, 3-10
    program shell (illustration), 3-7
    reading measurement, 3-10
    triggering devices, 3-9
    waiting for measurement, 3-9 to 3-10
NI-488 functions. *See also* auxiliary functions, wibic.
    one device per board, 3-2 to 3-3
    parallel polling, 7-10 to 7-12
    programming considerations
        advantages of using, 3-2
        board functions, 3-2 to 3-3
        device functions, 3-2
    serial polling, 7-7 to 7-8
    using in wibic
        examples, 5-1 to 5-4
        ibdev, 5-10 to 5-11
        ibfind, 5-10
        ibrd, 5-12
        ibwrt, 5-12
        syntax, 5-5 to 5-7
NI-488.2 applications, programming
    configuring instruments, 3-15
    finding all Listeners, 3-13
    flowchart of programming with routines, 3-12
    general steps and examples, 3-13 to 3-16
    identifying instruments, 3-13 to 3-14
    initialization, 3-13
    initializing instruments, 3-14
    items to include, 3-11

placing board offline, 3-16
processing of data, 3-16
program shell (illustration), 3-12
reading measurement, 3-16
triggering instruments, 3-15
waiting for measurement, 3-15 to 3-16
NI-488.2 DLL
choosing how to access, 3-1
requirements for direct access, 3-1
unloading or reloading NI-488.2 driver, 1-9
NI-488.2 routines. *See also* NI-488.2 applications, programming.
parallel polling, 7-12
programming considerations, 3-3
serial polling, 7-8 to 7-10
serial polling examples
using AllSpoll, 7-9 to 7-10
using FindRQS, 7-9
using in wibic
issuing set command before using, 5-13
Receive, 5-13 to 5-14
Send, 5-13
SendList, 5-13
wibic syntax, 5-8 to 5-9
NI-488.2 software, 1-6 to 1-9. *See also* application development; NI-488 functions; NI-488.2 routines.
C language files, 1-7
driver and driver utilities, 1-6 to 1-7
interaction with Windows, 1-8
reloading, 1-9
unloading, 1-9
Visual Basic files, 1-7
win 32S files, 1-8
nivdmad.386 device, 1-6
nivgpibd.386 device, 1-6
NRFD (not ready for data) line, 1-3
number syntax in wibic, 5-4

O

output options, GPIB Spy utility, 6-4

P

parallel polling, 7-10 to 7-12
    application example, 2-18 to 2-19
    implementing, 7-10 to 7-12
    setting duration of, 8-10
    using NI-488 functions, 7-10 to 7-11
    using NI-488.2 routines, 7-12
PPoll routine, 7-12
PPollConfig routine, 7-12
PPollUnconfig routine, 7-12
primary GPIB address
    definition, 1-2
    purpose, 8-7
    setting, 8-7
print (display the ASCII string) function, wibic, 5-17
problem solving. *See* debugging.
programming. *See* application development; debugging; GPIB programming techniques.

R

readme.txt file, 1-6
ReadStatusByte routine, 7-8
Receive routine
    reading measurements, 3-16
    using in wibic, 5-13 to 5-14
REM status word condition
    bit position, hex value, and type (table), 3-4
    description, A-4
REN (remote enable) line
    purpose (table), 1-3
    setting for automatic assertion, 8-9
repeat addressing
    communication errors, 4-6
    enabling, 8-11
repeat function n times (n*) function, wibic, 5-16
repeat previous function (!) function, wibic, 5-15
requesting service. *See* service requests.
routines. *See* NI-488.2 routines.
RQS status word condition
    bit position, hex value, and type (table), 3-4
    description, A-2 to A-3
running applications. *See* compiling, linking, and running applications.

S

secondary GPIB address
    definition, 1-2
    setting, 8-7
Send routine
    configuring instruments, 3-15
    using in wibic, 5-13
SendCmds function, 7-2
SendIFC routine, 3-13
SendList routine, 5-12
serial polling, 7-5 to 7-10
    application example using NI-488.2 routines, 2-16 to 2-17
    automatic serial polling, 7-6 to 7-7
        autopolling and interrupts, 7-7
        enabling, 8-9
        stuck SRQ state, 7-6 to 7-7
    service requests
        from IEEE 488 devices, 7-5
        from IEEE 488.2 devices, 7-6
    setting timeout value, 8-8
    SRQ and serial polling
        with NI-488 device functions, 7-7 to 7-8
        with NI-488.2 routines, 7-8 to 7-10
service requests. *See also* SRQ (service request) line.
    application examples, 2-10 to 2-13
    serial polling
        IEEE 488 devices, 7-5
        IEEE 488.2 devices, 7-6
    stuck SRQ state, 7-6 to 7-7
    Talker/Listener applications, 7-4
set command, 5-13
Set (select device or board) function, 5-15
setting up your system. *See* configuration.
setup utility, 1-6
software. *See* NI-488.2 software.
SPOLL status word condition
    bit position, hex value, and type (table), 3-4
    description, A-3
SRQ (service request) line. *See also* service requests.
    purpose (table), 1-3
    serial polling
        using NI-488 device functions, 7-7 to 7-8
        using NI-488.2 routines, 7-8 to 7-10
    stuck SRQ state, 7-6 to 7-7
SRQI status word condition
    bit position, hex value, and type (table), 3-4
    description, A-2

status word - ibsta, 3-3 to 3-4
ATN, A-4
CIC, A-4
CMPL, A-3
DCAS, 7-4, A-5
DTAS, 7-4, A-5
END, A-2
ERR, A-2
EVENT, A-3
LACS, 7-4, A-4
layout (table), 3-4
LOK, A-3
programming considerations, 3-3 to 3-4
purpose and use, 3-3 to 3-4
REM, A-4
RQS, A-2 to A-3
SPOLL, A-3
SRQI, A-2
TACS, 7-4, A-4
TIMO, A-2
wibic example, 5-9
string syntax in wibic, 5-5
stuck SRQ state, 7-6 to 7-7
System Controller
as Controller-in-Charge, 1-1
configuring, 8-9

T

TACS status word condition
bit position, hex value, and type (table), 3-4
description, A-4
monitoring for message from Controller, 7-4
talk address, setting, 1-2
Talker/Listener applications, 7-4 to 7-5
event queue, 7-5
requesting service, 7-5
waiting for messages from Controller, 7-4
Talkers, 1-1
technical support, C-1
terminate read on EOS, 8-8
termination methods, errors caused by, 4-6
termination of data transfers, 7-1 to 7-2
TestSRQ routine, 7-8
timeout value, setting, 8-7
serial poll timeouts, 8-8
timing errors, 4-6

TIMO status word condition
bit position, hex value, and type (table), 3-4
description, A-2
TNT4882C hardware, 7-2
*TRG command, 3-15
triggering devices, example, 2-4 to 2-5
troubleshooting. *See* debugging; wibic utility.
turn display off (-) function, wibic, 5-15
turn display on (+) function, wibic, 5-15

V

Visual Basic
compiling, linking, and running applications, 3-18
direct entry, 3-23
files available with NI-488.2 software, 1-7

W

wait function. *See* ibwait function.
WaitSRQ routine
conducting serial polls, 7-8
waiting for measurement, 3-15 to 3-16
wibconf utility
batch mode, 8-13 to 8-15
command pairs (table), 8-15
configuration file, 8-13 to 8-14
configuration options, 8-7 to 8-11
assert REN when SC, 8-9
base I/O address, 8-10
bus timing, 8-10
cable length for high speed, 8-10
DMA channel, 8-11
enable auto serial polling, 8-9
enable CIC protocol, 8-9
enable repeat addressing, 8-11
EOS byte, 8-8
GPIB-PCII/IIA mode switch, 8-11
interrupt jumper setting, 8-11
parallel poll duration, 8-10
primary GPIB address, 8-7
secondary GPIB address, 8-7
send EOI at end of write, 8-9
serial poll timeout, 8-8
set EOI with EOS on writes, 8-8
System Controller, 8-9

terminate read on EOS, 8-8
timeout setting, 8-7
type of compare on EOS, 8-8
use this GPIB interface, 8-10
default configurations, 8-12
exiting (<F9> or <Escape>), 8-12 to 8-13
checking for errors, 8-13
lower level device/board characteristics, 8-5 to 8-6
changing board or device (<Control-PageUp> and <Control-PageDown>), 8-6
changing characteristics (<PageUp>, <PageDown>, <Home>, or <End>), 8-6
Help option (<F1>), 8-6
Reset Value (<F6>), 8-6
Return to Map (<F9> of <Escape>), 8-6
screen illustration, 8-5
options for starting (table), 8-2
overview, 8-1
purpose, 1-7
reconfiguring GPIB driver, 4-5 to 4-6
starting, 8-1 to 8-2
upper level device map, 8-3 to 8-5
upper level device map options
device maps of boards, 8-4
(Dis)connect (<F5>), 8-4
Edit (<F8>), 8-5
Exit (<F9> or <Escape>), 8-5
Help (<F1>), 8-4
Rename (<F4>), 8-4
screen illustration, 8-3
wibic utility
auxiliary functions
! (repeat previous function), 5-15
$ (execute indirect file), 5-16
+ (turn display on), 5-15
- (turn display off), 5-15
buffer (set buffer display mode), 5-17
help (display help information), 5-15
n* (repeat function n times), 5-16
print (display the ASCII string), 5-17
set (select device or board), 5-15
table of functions, 5-14
checking for display errors, 4-4
communicating with devices, 3-5
count, 5-10
debugging applications, 4-4
error information, 5-9 to 5-10

NI-488 functions
examples, 5-1 to 5-4
ibdev, 5-10 to 5-12
ibfind, 5-10
ibrd, 5-12
ibwrt, 5-12
NI-488.2 routines
issuing set command before using, 5-13
Receive, 5-13 to 5-14
Send, 5-13
SendList, 5-13
overview, 5-1
programming considerations, 3-5
purpose, 1-7
status word, 5-9
syntax, 5-4 to 5-9
addresses, 5-5
board-level functions (table), 5-7
device-level functions (table), 5-6
NI-488 functions, 5-5 to 5-7
NI-488.2 routines, 5-8 to 5-9
numbers, 5-4
strings, 5-5
wibtest utility
diagnostic messages, 4-1 to 4-2
cable connections, 4-2
presence of board, 4-1 to 4-2
presence of driver, 4-1
purpose, 1-7
win 32S files, 1-8
Windows interaction with NI-488.2 software, 1-8
Windows Interface Bus Interactive Control utility. *See* wibic utility.